AD-A075 228 BATTELLE COLUMBUS LABS OH F/G 5/2
THE STATE-OF-THE-ART IN SOFTWARE ERROR DATA COLLECTION AND ANAL--ETC(U)
JAN 78 R THIBODEAU DAAG29-76-D-0100
UNCLASSIFIED NL

1 OF 2
AD-
A075228

DISTRIBUTION STATEMENT

Approved for public release. Distribution unlimited

This Technical Report has been reviewed and is approved.

Clarence Giese, Director
U. S. Army Institute for Research
in Management Information
and Computer Science

John R. Mitchell
Chief, Computer Science
Division, U. S. Army
Institute for Research in
Management Information
and Computer Science

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE *(and Subtitle)* (6) The State-of-the-Art in Software Error Data Collection and Analysis		5. TYPE OF REPORT & PERIOD COVERED (9) Final Reptis 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) (10) Robert Thibodeau		8. CONTRACT OR GRANT NUMBER(s) (15) DAAG29-76-D-0100
9. PERFORMING ORGANIZATION NAME AND ADDRESS General Research Corporation Economics and Resource Planning Operation 307 Wynn Dr., NW, Huntsville, AL 35805		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS U. S. Army Institute for Research in Management Information and Computer Science, 313 Calculator Bldg., GIT, Atlanta, Georgia 30332		12. REPORT DATE (11) 31 Jan 78 13. NUMBER OF PAGES 111
14. MONITORING AGENCY NAME & ADDRESS *(if different from Controlling Office)* (12) 117		15. SECURITY CLASS. *(of this report)* 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*
Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*
Same

18. SUPPLEMENTARY NOTES

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*
Data Acquisition, reliability, error analysis, quality

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*
Part of a comprehensive research program directed toward improving software reliability, this review of the state of the art in software error data collection and analysis describes the efforts of government agencies, educational institutions and private companies to collect and analyze software data. The author points out significant shortcomings in all present approaches to data collection which make the results of other organizations of marginal value to Army information system development programs.

over

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

407 080

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

#20. ABSTRACT (cont)
A proposed Army research program outlines a system of error data collection and analysis which the author believes will prove valuable in the effort to improve the quality of delivered software.

TECHNICAL DOCUMENTARY REPORT

U.S. ARMY COMPUTER SYSTEMS COMMAND

THE STATE-OF-THE-ART IN SOFTWARE ERROR DATA COLLECTION AND ANALYSIS

FINAL REPORT

Author: Robert Thibodeau

January 31, 1978

Prepared For:

ARMY INSTITUTE FOR RESEARCH IN MANAGEMENT INFORMATION
AND COMPUTER SCIENCE (AIRMICS)
Atlanta, Georgia 30332

Prepared By:

GENERAL RESEARCH CORPORATION
Economics and Resource Planning Operation
307 Wynn Drive, NW
Huntsville, Alabama 35805

CONTRACT DAAG29-76-C-0100/0598

ABSTRACT

Data that describe software errors and other data describing software and its development are an essential resource for research directed toward improving the quality of delivered software. This report describes the work that is presently being done by government agencies, educational institutions and private companies to collect and analyze software data. The findings indicate that the existing software data cannot be used directly by the US Army Computer Systems Command. A recommended research program is described that has the objective of improving the quality of delivered software. The recommendations consider data requirements, error classification, research topics and data analysis.

FORWARD

This document was prepared under a subcontract (D.O. No. 0598) to Contract Number DAAG29-76-D-0100 held by Battelle Columbus Laboratories. The report was written by General Research Corporation, Economics and Resource Planning Operation for the U.S. Army Institute for Research in Management Information and Computer Science (AIRMICS).

DISCLAIMER

DISPOSITION INSTRUCTIONS

ACKNOWLEDGEMENTS

Many persons took time to answer questions about their work in software data collection and analysis thereby contributing information valuable to the preparation of this document. We would like to acknowledge these contributions.

We would like to express our appreciation to Mr. John Mitchell of AIRMICS who guided our search for sources of error data and who reviewed our preliminary drafts of this report.

Mr. W. D. Flueckiger was the principal GRC reviewer for this document. It is GRC policy to have each document reviewed by a person not directly involved with its preparation. Others who reviewed the drafts of the report and made helpful comments were E. N. Dodson, S. L. Mitchell, and S. Saib.

TABLE OF CONTENTS

TABLE OF CONTENTS (Con')

1.0 SUMMARY

The objective of this study is to relate the present state-of-the-art in software error data collection and analysis to the US Army Computer Systems Command (USACSC) software reliability problem. This is accomplished by demonstrating how the availability of software data and the results of current studies can be utilized by the USACSC. A recommended research program is presented that is directed toward the reduction of the numbers of errors that occur in software delivered to the field.

The purpose of this review of the state-of-the-art is to investigate the process of collecting and analyzing software error data. However, the analysis of error data requires knowledge of the programming methods used, program characteristics, and other aspects of the software development. It is necessary, therefore, in many instances, to consider the collection and analyses of all kinds of software data. In some parts of this document a broad discussion of software data collection is included without making any distinctions between error and other software data.

A comprehensive literature search was the starting point for identifying current research related to the objective. Abstracts of studies and publications were used to compile a list of organizations and individuals who are active in the field of software data collection and analysis. Direct contacts were made to obtain firsthand information and to obtain referrals to other persons doing related work.

The state-of-the-art in software error data collection and analysis can be divided into five principal topics:

- Software Data Collection
- Error Classification
- The Causes of Errors
- Software Testing
- Software Reliability

Section 5.1 of this report discusses each of these topics in detail. The other topics included in the presentation of the state-of-the-art describe data collection practices and existing compilations of software data.

Our findings indicate that:

- Much of the existing software data was collected after the completions of the projects. This means that the data were compiled from available documentation supplemented by memory. Sometimes the persons who originated the documents were not available to assist in their interpretation.
- The principal sources of data were problem reports and modification notices which are used for configuration management or to ensure responses to problems reported from the field. They were not designed for research purposes.
- These data were supplemented in several studies by questionnaires given to project leaders and programmers.
- No standard taxonomy exists for classifying software errors. The classification schemes that were analyzed were found to be cumbersome and imprecise. The difficulties encountered by researchers in using the classification schemes were a major obstacle to making good use of the error data.
- Error classification was usually done after the fact using the problem reports. Often the person making the classification was not the same one preparing the report and in some cases it was not a person with direct knowledge of the software.

Several studies are documented that were designed to make use of existing software data to analyze or predict errors. Resources were provided that made it possible to screen the existing data and to collect accurate supplementary data. However, these projects were not successful in obtaining reliable relationships between errors and other variables.

Some of the researchers blamed their failure to achieve acceptable results on the quality of the available data. It is possible that the reason for failing to obtain reliable relationships was due to the analysis rather than the data. However, as is discussed in the report, certain deficiencies were found in the data and these must be removed before we can address the analysis.

Considering the experiences of these and other researchers, it seems unlikely that the USACSC can make use of presently available data other than as a subjective guide for planning its own future efforts.

Our recommendations to the USACSC include:

- A research program should be designed to investigate errors and the factors that contribute to them.
- The goal of the program should be the implementation at the USACSC of software development methods that are cost-effective.
- Reliance on data developed outside the USACSC is not recommended other than as a guide for initial planning.
- An initial period of data collection is a prerequisite for establishing a practical error classification system.
- Fundamental measures of software and related descriptors should be developed and validated with experimental evidence. These include measures of program size, complexity, etc.
- Explicit requirements for data items should be established before the data are collected. This should be accomplished by establishing a system of objectives and priorities. Hypotheses should be created that relate errors to their causes and other contributing factors. The data needed to test these hypothese should be collected by means of monitored procedures that place emphasis on data quality. This process is intended to produce a smaller volume of data than has been collected in similar projects in the past.

- An integrated structure that provides for systematic program development should be preferred over the development of specific techniques.
- Methods of automated data collection should be investigated.

The USACSC has the opportunity to conduct a software research program in an environment where many factors such as equipment and programming language are controlled. This by itself may make it possible to achieve the goal of quantifying software relationships that has eluded software engineers.

2.0 INTRODUCTION

2.1 Problem Background

The US Army Computer Systems Command (USACSC) develops and maintains standard software systems. The Command is responsible for logistics, inventory and stores, payroll, personnel, and other systems upon which the Army depends for its day-to-day operations.

The USACSC has a staff of over 500 computer systems development personnel located at Fort Belvoire, Virginia. The principal programming language is COBOL. The systems are developed and operated on IBM System 360 computer equipment.

The systems are developed and maintained at a single site. New ones are designed, tested and released to over 100 installations throughout the world. Errors that occur in systems that have been released to the field can be very difficult to diagnose and expensive to correct. It is therefore very desirable to eliminate as many errors as possible before they are released.

The reduction of errors in operational software systems can be accomplished in several ways. For example, improved testing procedures may significantly improve the quality of new programs by identifying more errors than is possible with existing methods. Another method for eliminating errors is to develop ways for creating input data sets that produce more thoroughly tested programs. Still another approach is to prevent errors from getting into the code by using better design and coding techniques. It may be that all these methods should be used.

Although much has been written about the value of different techniques for improving software quality, no systematic tests have been performed to substantiate the claims. Top-down design, structured coding, chief programmer teams and other software development concepts usually require changes

in organization and procedures to be implemented effectively. Supervisors and programmers must be trained and motivated to use the techniques. Changes in coding and testing procedures can have a significant effect on the use of computer resources. Therefore, it seems essential to test each method for improving software quality under controlled conditions. The objective should be to implement methods at the USACSC that have been demonstrated cost-effective in improving the quality of released software.

To develop new methods and to demonstrate that they satisfy these criteria it is necessary to have a source of information about software errors. These data are essential for testing the various hypotheses describing the expected effects of new procedures on the quality of released software.

Before designing a program of studies to develop methods of reducing errors in released software, it is prudent to consider two facts:

1. It is difficult and expensive to obtain good quality data describing software errors and the factors that affect them.
2. In order to investigate the many facets of program quality it is necessary to have as large and as varied a data base as possible.

Both of these considerations suggest the desirability of learning about the availability and usefulness of data collected by other researchers. The identification of sources of data would save the USACSC time and funds. If data relevant to the USACSC's needs are not available, the analysis of others' experiences will help the USACSC in designing and carrying out its own research program.

The USACSC presently utilizes certain forms and related procedures in the development and maintenance of its software systems. These documents should be considered as a possible source of software error data. The experience of other researchers in using similar sources should be investigated.

2.2 Objectives

Consistent with the objectives of the USACSC, this study is designed to learn who has been collecting software error data, what they have been doing with it, and whether their work has any value to the USACSC.

In each case we want to know the quality of the data. We want to know something about the process of developing the software and for discovering, classifying, and recording the errors. We want to know if the data is in computer-processable form and whether it is available to the government.

Finally, we want to develop recommendations for future research based on the needs of the USACSC and our study of the state-of-the-art in software error data collection and analysis.

3.0 PROCEDURE

In order to include all possible sources of software error data presently being collected, we began the survey with an extensive literature search.

The literature search results were analyzed to determine the research topics related to software error data collection and analysis, the sponsors, participating organizations and institutions, and key individuals.

Articles and reports were obtained that were believed relevant to our survey and these were analyzed to determine the needs for data, problems encountered in collecting and analyzing error data, and making recommendations for improvement.

The analysis of the literature was performed in conjunction with direct contacts. Individuals who had experience with software error data collection and analysis were able to assist in the interpretation of the studies and make recommendations for further research.

4.0 LITERATURE SEARCH AND DIRECT CONTACTS

4.1 Literature Search

Two automated literature searches and a bibliography on software reliability provided the starting point for identifying materials which would describe the extent of the state-of-the-art in software error data collection.

The Army's Redstone Scientific Information Center, Redstone Arsenal, Alabama completed the automated searches of two large bibligraphic files. One was a computerized list of abstracts maintained by the National Aeronautics and Space Administration (NASA). The other was a similar file maintained by the Defense Documentation Center (DDC) for the Department of Defense. (DoD). The user can obtain a search of either file for desired materials by selecting relevant key words.

The NASA file includes abstracts from the following sources.

- Aerospace Abstracts
- The Engineering Index
- NASA Reports
- NASA Contractor Reports
- NASA Briefs (Descriptions of research activities including agency and principal-investigator)
- NASA Contracts (Descriptions of contracts let but not completed)

The data base was searched using the following key words.

- Computer Programs
- Computer Systems Programs
- Data Acquisition
- Reliability
- Error Analysis
- Quality

The key words were used singly and in combinations. The universe of key words thus obtained numbered nearly 54,000. From this universe over 1400 abstracts were printed by the computer.

The abstract file of the Defense Documentation Center was searched in a similar manner and this produced a computer-generated list of about 1000 additional abstracts.

The NASA and DDC lists were then scanned for abstracts describing software studies dealing with the collection or use of software error data. A condensed list of about 200 abstracts was prepared. This list has been bound together and transmitted to the study Project Officer.

The NTIS has published an annotated bibliography[1] on software reliability. This document was searched for additional sources. None were found that were not already included in the first two sources.

This procedure resulted in the list of research topics, sponsors, organizations and individuals that appears to be representative of the state-of-the-art in software error data collection and analysis. This list is presented in Table 1.

4.2 Direct Contacts

Table 1 served as the starting point for making direct contacts wit' project officers and researchers. The purpose of the contacts was to obtain additional insight that would be helpful in interpreting the published results of the studies and in formulating recommendations for additional research.

We wanted to learn from persons who had been directly involved in error data collection and analysis, why they did what they did, what the results were, and what they would do to correct any problems or deficiencies. A questionnaire and a checklist were prepared to help guide the interviews. These are presented in Appendix A.

TABLE 1

SUMMARY OF RESEARCH

RESEARCH SPONSOR	INSTITUTION OR COMPANY CONDUCTING THE SEARCH	PRINCIPAL AUTHOR	TOPIC OR OBJECTIVE	ERROR DATA* COLLECTED (C) USED (U) NOT USED (N)
USAF/RADC	RADC	Sukert	Reliability modeling	U
	Raytheon	Willman	System reliability history	C
	SDC	Willmorth	SW data collection	N
	IBM	Baker	Data collection, project history	C
	IBM	Motley	Prediction of programming errors	U
	TRW	Brown	Impact of programming practices	N
	TRW	Thayer	SW reliability	C
	PINY	Shooman	Reliability measurement models	U
	PINY	Trivedi	Prediction of SW perf. params.	U
	PINY	Shooman	Effect of bug generation	U
	MITRE	Amory	Error classification method	N
USAF/SAMSO	Aerospace Corp.	Callender	Industrial practices	N
USA/BMDATC	Logicon		Error classification theory	N
	Logicon	Lambert	Reliable SW study	N
USA/Frankford	SofTech	Goodenough	Test data selection	U
NASA	Aerospace Corp.	Hecht	Measurement SW reliability	U
NADC	NPGS	Bradley	Structure and errors	C
SPONSORS NOT IDENTIFIED	Aerospace Corp.	Hecht	SW-Hardware reliability	U
	Aerospace Corp.	Reifer	Test tool overview	N
	Fujitsu Ltd.	Akiyama	Prediction of SW bugs	C
	IBM	Belady	Large program development	U
	IBM	Endres	Errors and causes	U
	Independent Consult.	Gilb	Unreliability	N
	Independent Consult.	Walsh	Structured testing	N
	Independent Consult.	Yourdon	Reliability measurements	U
	Inst. for Adv. Tech.	Ogdin	Reliability	U
	Logicon	Rubey	SW validation	U
	McDonnell Douglas	Moranda	Predicting SW reliability	C
	Naval Postgrad. School	Schneidewind	Error processes	U
	PINY	Shooman	Programming errors	U
	RAND	Boehm	SW development	N
	Swedish Nat. Def. Inst.	Palme	Languages for reliable SW	N
	SEL	Hausen	Measuring reliability	C
	Tracor	Sontz	Quality assurance	C
	USA/SAFEGUARD	Dickson	Reliability analysis	U
	USAF/Ogden	Shelley	SW reliability	N
	USN/NSRDC	Culpepper	Reliable engineering SW	N
	Voest-Alpine	Kopetz	Error detection	C

*The enties indicate whether the reported research included the collection of new error data, used existing error data, or did not required the use of error data.

The questionnaire and checklist were designed with the idea that detailed information describing existing software data sets would be available. Our contacts quickly indicated that this assumption was not justified. The forms proved to be awkward and hindered the exploration of information that would be of value in forming an understanding of the state-of-the-art in software error data collection.

The questionnaire and checklist were abandoned in favor of a free-form interview designed to obtain referrals to other researchers and sources of data and to learn firsthand experiences with data collection whenever possible.

The forms are presented in Appendix A because we believe they represent a useful list of relevant descriptors and important information that describe the attributes of a good software error data base. Reference to the lists may aid the construction of future inquiries about data base attibutes.

4.3 Descriptions of Relevant Studies

After completing the review of the available reports and articles, we believe that the state-of-the-art is well represented in this report.

Appendix B contains summaries of some relevant studies that were obtained from the survey. Included in the information presented is:

- The research topic
- The sponsor
- The participants
- Their organizational affiliations
- The use and source of error data

The summaries were prepared for articles that describe important aspects of the different areas related to software data collection and analysis. The articles described in Appendix B together with the information obtained from the direct contacts are the basis for the next two sections.

5.0 ANALYSIS OF THE STATE-OF-THE-ART

As we described in the preceding section, our approach was to review as much published information as possible in order to get a basic understanding of the scope and content of the state-of-the-art in software error data collection and analysis. Direct contacts with researchers were made to confirm our interpretation of their work and to get their suggestions regarding further research requirements. The purpose of this section is to summarize these findings in a way that will be helpful to the USACSC in constructing a research program to improve software quality.

The presentation of the state-of-the-art is divided into three principal topics describing:

- What is being done in error analysis,
- how data collection is accomplished, and
- what error data exists.

To describe what is being done in error analysis, the first subsection, Software Error Research Topics, presents five aspects of error research:

- Software Data Collection
- Error Classification
- Problems, Errors and Causes
- Software Testing
- Reliability

We believe these topics represent the bulk of the research presently being done. They also describe important areas of consideration for USACSC in the design of its own research program. Finally, analysis of these topics can indicate the applicability of the existing research and its findings to the USACSC's research program.

We have directed our attention to areas of software research that require error data. The subsection, Data Collection Practices, describes how data to support research activities is being obtained. It is divided into four topics:

- Software Problem Reports and Modification Notices
- Error Interpretation and Classification
- Collection of Supporting Information
- Automated Data Collection

These topics describe the principal sources of data for the studies included in our search. They also discuss problems encountered when attempting to collect data and some methods for automating the data collection process.

The subsection, Descriptions of Existing Software Data Bases, describes the data collections undertaken by:

- USAF Rome Air Development Center (RADC)
- USA Ballistic Missile Defense Systems Command (BMDSCOM)
- USAF Data Systems Design Center (DSDC)

Contacts were made with persons with direct knowledge of these data collections. Their responses to our questions and their additional comments helped us to develop our recommendations.

5.1 Software Error Research Topics

It is necessary to consider the research being conducted in software quality before describing the data collection that is going on. This is because we will gain a better understanding of the usefulness of the existing error data if we know why it was collected and how it is being used.

The following paragraphs describe some of the current research topics that relate to software error data. Each topic is developed along lines that have a bearing on problems faced by USACSC. It should then be possible to draw conclusions about the applicability of the data to the needs of USACSC.

5.1.1 Software Data Collection

The analysis of error data requires knowledge of the programming methods used, program characteristics, and other aspects of the software development. It is necessary, therefore, in many instances, to consider the collection and analysis of all kinds of software data. Much of the following material considers the problems of collecting software data without regard to whether it is error data or other descriptive information. Since the underlying problems are the same, we don't believe that any confusion will result.

The need to conduct formal studies of the software development process has been recognized for some time. The rapid growth in the use of software as integral parts of large systems made it imperative to learn the factors that affect the cost and quality of the final product. This section gives a historical perspective on software data collection and briefly describes how software data collection problems have influenced attempts to study software quality.

One of the first large scale attempts to measure the factors that affect the software development process was undertaken by the Air Force in the middle sixties. A series of studies was completed by Systems Development Corporation [4,5,6,7] which had as its objective the quantification of program characteristics, personnel factors, and project management techniques on the cost of software. Over one hundred factors were selected for testing using actual project and software data.

The initial attempts at establishing predictive relationships were made using in-house data describing relatively small programs. These were done more to develop the analytical procedure than to obtain useful relationships. Subsequently, the data collection was extended to include about 170 programs. The data set included both large and small programs of different types.

A rather careful and sophisticated analysis failed to establish predictive relationships even though the best available data were used. The conclusion was that the data were not of sufficient quality.

SDC recommended that procedures be established for the collection of software data for future research[8]. The recommendation was not implemented.

Later, studies were undertaken for the Air Force by Planning Research Corporation[9] and General Research Corporation[10]. These studies attempted to quantify the factors affecting software development costs considering the state-of-the-art in 1970 and 1976 respectively. In both cases failure to obtain useable relationships was attributed to the quality and quantity of available data. GRC proposed a comprehensive data set describing project and software characteristics to be collected during future Air Force software development projects.

Recently, the Air Force contracted with SDC to conduct an analysis of the problems of software data collection and to study the feasibility of establishing a repository of software data[11]. SDC studied a number of aspects of the data collection problem including cost, resistance of software developers, interference in the software development process, and methods of storage and access. The study will serve as the basis for creating a software data repository at Rome Air Development Center (RADC).

A number of software development contractors have been given assignments by RADC to create data sets for a series of studies sponsored by RADC. TRW, Raytheon, IBM, Boeing, and Draper Labs have been commissioned to do this. Several of these projects are described in the abstracts.

The usual procedure followed by these organizations to obtain the required software error and development data was to:

1. Obtain problem descriptions from problem reports and modification notices.
2. Assign teams of programmers to identify the errors causing the problems.
3. Classify the errors according to a taxonomy developed by TRW.
4. Collect certain supporting data describing the software characteristics and development environment.

When the data developed according to this procedure were used in studies, it was apparent that their poor quality negatively affected the results. The following factors were believed to affect the quality of the data.

- The descriptions of the classification criteria were not clear.
- The error classifications were not necessarily made by the same person reporting the problems.
- The intensity or extent of testing was not known.
- Data that are subject to change such as program size were not recorded at the same time that the problems were experienced.

Some of the most sophisticated software systems ever developed have been associated with the Army's ballistic missile defense systems. These real-time systems resident on state-of-the-art hardware have focused attention on the need for improved methods for software development.

In support of its studies of software engineering the Ballistic Missile Defense Systems Command (BMDSCOM) collected software development and reliability data for several years.[17]

The Army experience is similar to that of the Air Force. Data obtained from different contractors is not compatible and there is reason to suspect the quality of the reported data. Much of the data was obtained

after the fact with the usual inaccuracies that result. Furthermore, there is no evidence that the data obtained from the ballistic missile defense environment and its associated contractors is applicable to the USACSC environment or any other.

Our study of the experience in the collection of data related to the software development process indicates that the same mistakes have been repeated over and over again. In 1967 after spending much time and resources trying to develop useful relationships from poor quality data, Nelson* made the following observation:

> "All the data used from both the statistical analysis and the literature were data of opportunity, i.e., we took what we were able to get in the time available. Hard data on the costs of computer programming..... are scarce commodities both in computer programming organizations and in the published literature. Few numerical data are recorded; fewer yet are recorded under 'controlled' conditions, and still fewer are suitable for generalization to other situations.....The respondents to the questionnaire were under no obligation to assure completeness and accuracy even when data were readily available. Because they were suspect, some of the data collected were rejected prior to the analysis. But even those data used in the analysis are likely to have a variation in reliability...."

Unfortunately, more than ten years later the statement is not only as true as it has ever been, but software researchers have attempted to repeat SDC's studies with the same poor results.

In 1973 Boehm** wrote:

> "One of the major problems the CCIP-85 study found was the dearth of hard data available on software efforts which would allow us to analyze the nature of software problems, to convince people unfamiliar with software that the problems were significant, or to get clues on how best to improve the situation. Not having such a data base forces us to rely on intuition when making critical decisions on software, and....software phenomena tend to be counterintuitive. Given the magnitude of the risks of basing major software decisions on fallible intuition,....it is surprising how little effort has gone into endeavors to collect and analyze such data."

---

* Ref 6, p 11

** Ref 23, p 59

The need for automated data collection, consistent definitions and a more manageable error classification system are some of the major needs demonstrated by the current experience.

It seems safe to conclude that if we intend to develop quantitative relationships between software errors and their causes, we are going to have to start at the beginning with a carefully designed program of research that avoids the pitfalls that have been described and ignored too many times already.

5.1.2 Error Classification

A software error is any deficiency in the design or implementation of a computer program or its related documentation that causes the results of the program execution to be different from those desired by the user of the software. This definition, therefore, includes errors or omissions in the program specifications as well as the coding of the program.

The study of software errors requires them to be separated according to their attributes. This is the first step in understanding what causes them and, subsequently, how they may be prevented. The need for a practical error classification is important and, since it applies to nearly all areas of software research, it deserves to be treated as a separate topic.

Although several studies have produced recommended classification methods for reporting software error and other data,[2,8,10,11] the majority of the available data is classified according to schemes that were developed for the purposes of the particular studies. For example, the studies described in references 3,4,9,12 and 13 used data that were classified according to schemes that were constructed specifically for these studies. As a result, there are almost as many classification systems as there are error studies. Even where a classification scheme was specified, as in the case of the RADC studies,[14, 15] awkwardness in design or incomplete documentation led to the introduction of new categories or inconsistent

use of the classifications. This has made it unlikely that data from one environment can be used to draw inferences about another environment. It also prevents data from different sources from being combined to provide a more extensive set.

Taken as a group the existing data classifications include the following deficiencies:

- Language Dependency. The error classes are defined in terms of specific language elements that do not have meaning for different languages. e.g.
  - FORTRAN FORMAT error
  - Improper termination of a DO loop (which limits the applicability of the definition to FORTRAN)

- Hardware/Operating System Dependency. The class definitions describe operations or equipment characteristics that are unique to a given configuration or system. e.g.
  - Failure to include access key on RUN card

- Ambiguity. The class description does not adequately differentiate between two similar conditions or does not indicate how to classify errors which could apply equally well to more than one category. e.g.
  - Total number of entries computed incorrectly
  - Index computation error
  - Wrong equation or convention used
  - Results of arithmetic calculation inaccurate/not as expected
  - Mixed mode arithmetic error

- Duplication. Two categories describe the same error condition. Usually this occurs when one category is really a subset of another. e.g.
  - Index computation error
  - Index not checked

- Insufficient Scope. Users find it necessary to add classes to cover errors apparently not covered in the scheme.

- Open Endedness. No limits are imposed on the addition of categories resulting in a proliferation of definitions.

- Impractically Large Numbers of Categories. The numbers of categories get so large that users must scan long lists. Often this situation is accompanied by ambiguity and duplication resulting in inconsistent classifications of errors.

Any error classification method must have one important attribute. It must make distinctions between problems observed during execution of a software system and the causes of those problems. The problems are manifestations that something is wrong with the software; they are symptoms of errors and not errors.

An example of a problem is a misplaced heading on a report. A more serious problem would be a wrong total for a column of numbers. The error in the first example might be miskeying a literal expression when coding the program. The error in the second example may be an improperly written arithmetic expression or an incorrect looping procedure. The latter error may actually consist of several errors.

It is essential that the classification system allow the researcher to distinguish between the problem and the error that caused it.

A great difficulty in designing a practical classification system is that sometimes a problem is caused by more than one error.

In some cases it may be difficult to assign an error to a class even if the error is known. For example, if a table is exceeded is the error that the subscript is too large or that the table is too small? Or, is the error omitting a check on the size of the subscript?

Another situation occurs when a problem observed in one module is caused by an error in another. The classification system and the procedure for recording supporting data should properly describe this condition. This is particularly important when trying to establish cause and effect relationships.

We should add one more dimension to the analysis of classification systems. A distinction has been drawn between problems and errors in software. Errors are the causes of problems, but we are really interested in controlling the causes of errors.

When reading the subsequent discussions it is important to note that we consider the description of the causes of errors to be outside the scope of the classification system. Errors are caused by many factors most of which are too poorly understood to be included in a practical classification system. The recommended approach is to construct hypotheses concerning the causes of errors and then to test these as part of a study.

Errors in software are caused by the same things that contributed to most human failings: lack of preparation, inattention, lack of motivation, fatigue, incompetence, insufficient time or resources, misunderstanding, etc. We can reduce errors by demonstrating that certain procedures or practices are associated with fewer errors because they compensate for human failings. We can accomplish this by first developing a classification system that properly describes errors in operating software. Then t' relationships between these errors and variables describing program characteristics, programming techniques, management methods, etc. are stated as hypotheses which are tested and accepted or rejected.

Two error classification systems should be mentioned. One was completed by MITRE Corporation[2] for RADC as part of a broad study of software quality. The other was accomplished by TRW,[3] in a series of studies ending in the Software Reliability Study which was also commissioned by RADC.

The MITRE system is not specific. It is more a methodology to be followed in establishing a practical classification scheme. One of MITRE's recommendations is that any classification system be open-ended so that it can adapt to the changing needs of researchers. It also permits the creation of non-exclusive definitions.

The TRW system went through a series of changes before it was used for the Reliability Study. The last version is said to be consistent with the MITRE scheme.

The TRW classification system is the closest thing to a standard. This is mainly because RADC is requiring several of its contractors to use it. However, the system has several shortcomings:

- It has more than 160 subcategories which, we believe, makes it cumbersome to use
- It is language dependent (JOVIAL-J4, CENTRAN)
- Errors and symptoms are intermixed
- Some of the subcategories overlap
- A detailed, written description of the subcategory definitions is not available.

These observations should be taken into consideration in any classification scheme considered for use by USACSC.

5.1.3 Problems, Errors and Causes

There are many factors that are related to errors or contribute to a climate in which errors are more likely to occur without actually being the causes of errors. These factors include such things as:

- Program size
- Program complexity
- Numbers of interactions among system modules
- Time and capacity constraints
- Programming language

We want to know the degree to which these are related to the production of errors. This will allow the development of hypotheses for methods to control errors. For example, top down design and structured coding would be methods of attacking the first three factors. Methods for defining and testing specifications would address the fourth factor and new or modified languages would deal with the last one.

A major difficulty in obtaining quantitative relationships between errors and contributing factors is the need for a large amount of data collected under controlled conditions. Both the number of different data items and the controlled conditions cause problems.

The number of data items becomes large for many studies because there is very little theoretical guidance available to aid us in defining data needs. For example, assume we want to include a measure of program size as a possible predictor of programming errors. The immediate question is: which is the better predictor of the number of errors in a program, the size of the source program or the size of the executable program? One could argue that the source program is the one written by the programmer and that lines of source code are indicative of the probability of incurring errors. The greater the number of lines of source code, the greater is the possibility of including an error. One could also say that the size of the program in its executable form is a better indicator of complexity because it includes all the library routines and other code to permit the execution of the intended process. Also, because some higher order language statements generate large amounts of executable code, the size of the executable program is a better indicator of the potential for error than the size of the source program. This reasons that a source statement generating many executable instructions is more likely to be associated with an error because there are more things that must enter into the generation of the executable code.

Other considerations of program size include:

- Lines of code written versus code taken from other programs.
- Library or other existing modules that are more thoroughly tested than new lines of code.
- Lines of code added or deleted in response to errors.

The last item requires program size to be a variable and the error process a dynamic one reflecting changes in program characteristics as problems are detected and corrected.

It is easy to see that as we consider other members of the list of error factors that similar difficulties exist when we try to identify data items to be collected.

Researchers have approached the problem of identifying data items in several ways. Some have tried to collect as many items as possible. This almost invariably reduces the quality of the data. Others have made arbitrary decisions at the beginning. The danger in this approach is that if the wrong items are selected it will be impossible to establish the desired relationships.

Very often the decision on which data items to collect is based on the ability to collect it. If the data are being collected retroactively many of the items do not exist. This has happened in many studies. Other problems are cost and facilities. Some of the items described in connection with program size are best obtained automatically during the testing process. However, if the tools needed to do this do not exist, the cost may be too great to collect it manually.

Our conclusion from this discussion is that it is very difficult to identify and accurately measure all the factors that influence the occurrence of errors in software. One way to reduce the problem is to eliminate some of the variables. For example, after failing to explain some of the results obtained using some experimental data to predict numbers of errors in programs, one reseacher[16] recommended that only programs of the same size be considered in a subsequent study. This concept has general applicability. The more variables that do not change among programs about which we are trying to draw inferences, the better are the chances of obtaining relationships between the production of errors and the remaining variables. This approach suggests two important facts that have implications for USACSC:

1. Without proper controls it is not wise to use data or results from one environment in another.
2. If studies are undertaken in a closed environment such as at USACSC many of the variables are automatically controlled.

Data obtained from systems developed at the USACSC should be good for establishing relationships between errors and factors that contribute to making them because the influences of some variables that are difficult to measure can be minimized. Since the development occurs in a single environment, things such as management methods, programming conventions, programming languages, computer hardware/software, etc. are similar. Therefore, the influences of these items are much less than they are for data taken from other organizations. Once relationships are established for the variables that do change at the USACSC, management methods, programming conventions, etc. can be changed under controlled conditions. This improves the likelihood of obtaining reliable measures of their effects.

One recently completed study may illustrate some of the problems in obtaining good software data. This case is not presented because it is bad or good, but because it is current and it uses what is probably some of the best data available.

The objective of the study was to develop equations which could be used to predict the numbers of errors in a software system using program characteristics and programmer variables. As it turned out, the study produced as much insight into the importance of reliable data for performing software error analyses as into the relationships between the predictor variables and errors.

Two data sets were provided for the study by the sponsor. They represented large command and control systems and they were collected for purposes other than the error prediction study.

One set was composed of data from three different projects. The programs were written in CENTRAN and 58 program characteristics had been obtained by an automated source code scanner. Error data was available only for the system test and integration phase.

The other data set was composed of eight subsystems of the same system. The programming language was JOVIAL-J4 and 16 program characteristics had been obtained by automated source code scanning. A different scanning program developed by a different contractor from that of the first data set was used. In addition to the program characteristics, variables describing programmer ability and workload were made available.

The report described the following problems regarding the quality of the available data.

- No quantification of the testing effort was available. This made it impossible to know if the different programs were tested to the same extent or using the same techniques.
- The errors included design-related problems (64% of the total errors) which had to be eliminated before the analyses could be made.
- The measurements of program characteristics were obtained at times other than when the errors occurred (up to three years in some cases). Grouped values were used for the program variables in order to minimize the effects of any variations associated with program changes. However, the effect on the results of the difference in time between measurements had to be considered unknown.
- The individuals who assigned the error categories were not the ones who recorded the problems on the problem reports. This can lead to misclassification if the problem descriptions are too succinct or are imprecise
- It was not clear in some instances why the predictor variables were collected or how they were expected to contribute to the understanding of programming errors.

- The lack of clear definitions for the predictor variables made it impossible to make any comparisons between the two samples much less other projects or programming languages.

- The analysis was limited by the lack of adequate definitions for the error classifications.

- There was too much variation among the measurements of variables within the data sets. This prevented the establishment of statisically valid relationships between the independent and dependent variables.

The results of this study indicate that we still have a long way to go toward the acquisition of reliable software error data much less establish relationships between errors and their causes.

5.1.4 Software Testing

The development of methods for testing software complements the research into the causes of software errors. Whereas the former seeks to eliminate errors from completed software, the latter attempts to avoid them in the first place.

Research into software testing methods is very broad and includes the entire software development cycle. It includes methods for examining specifications for consistency and completeness as well as testing code for compliance with standards. It includes aids for presenting the details for computational process in an easily checkable form. The methods may be manual, semi-automated or fully automated. Some test methods assist in the preparation of test data and others indicate how extensively the program paths have been traversed.

The development of software testing methods usually results from logical concepts rather than from empirical studies. New test techniques are

not derived from observations of error data. They are constructed using the researcher's ingenuity and his knowledge of logic and computer system operations. New techniques are validated by comparing the test results obtained from specific programs with other testing methods. Therefore, software testing methods research does not create a need for a large collection of error data.

The development of error seeding techniques is an example of an area of research in software testing methods where it is useful to have error data. Errors are deliberately introduced into computer programs and then testing methods are used to find them. This technique is useful both as a validation of a test procedure and as a basis for drawing inferences about the number of errors remaining in the program. The error seeding is more representative of the operating environment if the distributions of the errors by type are obtained from error data taken from programs in an operating environment.

5.1.5 Reliability

The final topic to be considered under software error research is reliability. Early computing devices were very complex systems made up of many electronic and mechanical components. The numbers of fragile components and the numbers of interconnections were large. The reliability of these systems was a prime consideration in many applications. Sometimes the time required to complete a calculation was of the order of the mean-time-to-failure (MTTF) of the computer. The reliability of the hardware components was an important field of study for computer hardware designers.

Microelectronics and other advances in electronics as well as reliability engineering techniques have so increased the reliability of computer hardware that when a computing system fails to perform properly, it is more likely to be caused by a fault in the software than the hardware.

Viewing a computing system as both hardware and software components either of which may cause a failure, it seems natural to apply the techniques of hardware reliability engineering to the prediction of software faults.

Using this concept, a software system is considered the same as a physical component. Therefore, in any given period of time there is some predictable probability that it will fail to perform properly.

The software reliability engineer applies the methods of hardware reliability estimating to software. A software system is operated under controlled conditions until a failure occurs. The results obtained from a series of observations are used to make estimates of the amount of time that the system can be expected to function in an operational environment before a failure occurs. Sometimes estimates are made of the total number of errors remaining and even of the probabilities of errors of different severities.

Critics of the software reliability concept say that there is no basis for treating hardware and software the same. They contend that physical components grow old, wear out, are of varying quality, etc. and that these qualities can be studied under laboratory conditions. The results of controlled tests can be used to obtain empirical factors with which to predict how systems made up of the components will perform under operating conditions. Software, on the other hand, does not deteriorate with age. The time until an error occurs is not dependent upon how long the testing is conducted, but on the completeness of the testing.

Because the measure of reliability or rather the measure of unreliability is the failure of the software to perform properly, the need for useful data with which to test hypotheses is relatively easy to satisfy. If the system fails to perform for any reason attributable to the software, an event has occurred which has validity for the reliability researcher. It is probably the availability of such error data that has caused reliability research to be so popular. It clearly represents the largest single type of research activity.

The two most commonly used measures of reliability are mean time to failure (MTTF) and failure rate. Other measures are errors per line of code and total errors. Current studies of reliability are devoted to

developing methods for measuring, estimating and predicting these quantities. Usually, the reliability researcher is not concerned with why the failures occur. Most reliability research is devoted to the creation of mathematical models that predict the reliability measures from program characteristics or test measurements.

The data requirements for reliability studies usually consist of counts of the occurrences of errors as a function of some time base. These data can be obtained from problem reports without requiring any additional data collection. It is necessary, however, to do some analysis of the error reports to eliminate duplications and to distinguish between software errors and such items as hardware failures and suggestions for improvement.

Problem reports and modification notices are routinely kept for configuration management purposes during the development of large software systems. This is why much of the reliability work reflects the experience developing command and control systems, large real time systems, or operating systems. Clearly the availability of data has influenced the direction and form of much of the reliability research.

In some studies of software reliability, actual data is used only to provide some insight or to justify the form of the hypothesis or model. For example, the time distribution of error detection during checkout may suggest a mathematical function that can be incorporated into the model. Or, it may suggest a hypothesis regarding the relationship between errors found and errors remaining that can form the basis for a statistical prediction method.

Reliability researchers may generate their own data. For example, in one study involving seeding errors the researcher had some small programs written and then tested them. The reliability model was then used to compare the predicted errors with those intentionally placed in the program.

In spite of the attention that software reliability models have received, no satisfactory predictors of program reliability have been developed. It seems likely that the critics of the approach may be correct and that software relaibility is significantly different from hardware reliability. At the very least, any predictions about the probability of a program containing errors must be based on some measure of how thoroughly the program has been tested rather than how long it was tested or executed.

One major sponsor of research on reliability models said in a conversation that his agency is discontinuing further research in this technique.

5.2 Data Collection Practices

5.2.1 Software Problem Reports and Modification Notices

As was indicated above, problem reports and modication notices have been the sources of the majority of software error data. These documents are widely used to control the development of large software systems and to ensure effective responses to problems occurring in software under maintenance. Appendix C contains a number of examples taken from the materials developed from the literature survey.

Appendix C shows that while details vary among the documents, they all contain the same basic information.

1. Program in which problem was observed
2. Description of the problem
3. Date problem was observed
4. Acknowledgement of the problem by the responsible entity
5. Disposition of the problem

Notice that for most reliability models, items 2 and 3 suffice. They only require an indication that an error has occurred and when. Care must be taken, however, to sufficiently screen the data to eliminate

duplicate reports of the same error,* reports resulting from improper use of the system, and suggestions for improving the system which are often reported as errors.

Since the primary purpose of these forms is to exercise management control and not to provide research information, there is little motivation to insist on precision or consistency in describing the problems. This makes the job of editing the documents for research rather difficult and reduces the quality of the data.

Several carefully executed efforts to collect error data,[3,12,14,15,16,18] have used the problem reports and modification notices as starting points and have supplemented the contents with additional data.

Five companies were given contracts to extract data from these forms, edit it for consistency, completeness and credibility, and then supplement it with additional data. The amount of additional data differed according to the needs of the studies being conducted.

In most cases the problem reports were studied by the programmers responsible for correcting the problems. If the responsible programmer was not available, a knowledgeable substitute performed the task. The errors causing the problems were identified and classified according to the TRW taxonomy. All of this information was reduced to machine-processable form for analysis.

---

*Duplicate reporting is very common for systems that are supported at a central facility such as the USACSC. Also, the problems may appear to be different because of the different ways of describing the same problem. One other obstacle is that different data sets may cause the same error to appear as different problems.

Some projects developed data describing program characteristics, project management descriptors, and even programmer ability indicators to supplement the error data. Most of this data was obtained long after the programs were completed. The responses were based on available reports, notes, or recall. In some cases the programs were run through automated analyzers that compute various characteristics.[3,16] It seems likely that data obtained after program completion are of lesser quality than if they had been obtained under controlled circumstances concurrently with the development and testing of the programs. Some of the measures were highly subjective (e.g., programmer ability). Others such as the error classifications were inconsistently applied, and the reliance on memory for some was not good. The processing of the source programs by analyzers was accomplished using versions of the programs which were not necessarily the same as the ones that contained the errors.

These deficiencies may have contributed to the poor results obtained using the data. For example, one study[19] had to create four data sets from one original data set in order to cover all the possible interpretations resulting from inconsistencies and omissions. Another researcher specifically cited the poor quality of the data as a factor to be considered in interpreting the results of his research.

These experiences resulted not from a casual data collection effort, but from a serious attempt to extract good quality software data by utilizing available sources.

It seems evident that if problem reports and modification notices are to be used to generate data for software error research, some of the deficiencies described above will have to be eliminated.

5.2.2 Error Interpretation and Classification

In almost all of the studies of software errors included in this survey, the interpretation and classification of errors was completed after a period of time had elapsed from when the problem was observed.* The classification was usually done by a person other than the one who observed the problem. It very often was done by a person who had no direct knowledge of the affected code. This practice leads to the very real possibility that the identified error was not the cause of the problem. It also contributes to a possible misinterpretation and misclassification of the error.

The more difficult practice to control, however, is the proliferation of error classification systems. In each of several studies included in the survey, systems for classifying errors were developed for the studies without apparently considering the adoption of previously used systems. Failure to report error research using a standard terminology has greatly limited the ability to compare results. Everyone has to start from the *beginning without being* able to build on the work done by others.

Most of the classification schemes were not developed systematically. After a general list of error categories was created, the analysts were permitted to add categories whenever a particular error did not seem to fit. This is like trying to build a house one room at a time. Structural strength and unity are lost. Inaccuracies occur and it becomes difficult for anyone to see how the different parts of the system contribute to the understanding of the error process.

Definitions have not been written in detail. As a result different persons use the categories according to their own interpretations.

---

* We are excluding studies involving small programs that were created specifically for the purpose of error analyses.

As was stated in Section 5.1.2, RADC has been working on the development of a classification system for software errors. In that section we pointed out some of the problems that exist in the system. RADC is aware of the need to improve the system and has taken steps to do so. Since the RADC scheme reflects a formal structure and because it has evolved through a series of actual applications to real data, there is a strong possibility it may provide an industry standard. At any rate the RADC experience should not be ignored when considering the construction of a new system.

5.2.3 Collection of Supporting Information

The study described in Section 5.1.3 is a good example of the existing experience in collecting data which is needed to analyze the causes of errors and to develop methods of predicting them. We can summarize the current status as follows:

- No basis exists for identifying a practical set of data items so that data collection costs can be kept to an acceptable level.
- Error, project, program, and resource data is collected retrospectively rather than during the software development and testing.
- Definitions of supporting data are not well documented which leads to ambiguity of interpretation.
- Allowances are not made for funding and staffing the data collection function as part of the development effort. These activities are funded separately usually by organizations different from the one that sponsor the development.
- Few resources are available to software developers for the automatic collection, storage, and retrieval of supporting data.

5.2.4 Automated Data Collection

Automated collection of software data represents one of the most promising ways to satisfy the requirements of concurrency, minimum interruption of the development process, consistent recording, and reasonable cost.

Prototype programs exist that can perform some of the required functions. Others can be developed to make a full range of data collection functions for software development projects. SDC in their survey of data collection practices[11] described the following automated data collection tools.

- IMPACT/FACE
- SIMON

The USAF Data Systems Design Center uses an automated software project reporting and analysis system called PARMIS.

Language scanners such as those described in Section 5.1.3 are also available.

5.3 Descriptions of Existing Software Data Bases

5.3.1 USAF Rome Air Development Center (RADC)

The mission of the Rome Air Development Center (RADC) includes software engineering research. In 1975 it was determined that such investigations would be greatly enhanced by a software data repository. The purpose of the repository would be to make available in a single facility a large body of data describing the software development process, software reliability and other characteristics.

The repository would be valuable as a research resource because uniform standards would be imposed during the collection and tabulation

of the data. Standard terminology and classifications would allow the combination of data from many sources. Therefore the universe of data available for analyzing any given area of software research would be greater than it is now. The present software data is characterized by poor or unknown quality control, differing definitions and unique systems of classification.

A baseline version of the repository is presently being used by the RADC staff and some of its contractors. It is not yet available to outside researchers. Plans are being finalized for the Data and Analysis Center for Software (DACS). This program, which will begin shortly, will make software data available on an incremental basis until the final system is operational in about three years.

5.3.1.1 Objectives of the RADC Data and Analysis Center for Software (DACS)

The DACS will provide data to the software research community on all phases of software development and will include many types of applications. Its primary objectives are to support USAF software research in the following areas:

- Reliability
- Language development
- Program planning and estimating
- Program management

5.3.1.2 Description of the DACS Operation

RADC is promoting the recognition of software data collection as an integral part of the software development process. Their objective is to include these data items as deliverables under Air Force contracts. Ideally, a full range of development environment, project organization, resource utilization, and software characteristics would be recorded during the conduct of the project.

Figure 1 describes the major categories of data to be included in the data base.

The sources for the data repository will include both questionnaires, existing project forms, and automated collection. The preference is for automated collection using library software to capture program characteristics and resource utilization. Where automated project reporting exists, it will be possible to capture this data as well.

The environmental and project organization data will be obtained from questionnaires being developed by RADC.

The repository will be maintained by an automated system.

Some possible features of the DACS include:

- Remote data entry
- Online data base access by telephone
- Automatic project management reports

5.3.1.3 The Present Status of the DACS

In 1975 RADC awarded five data collection contracts. These contracts were intended to provide data that would be useful in research on reliability models. The contracts were given to companies who had completed the development of software systems that were of interest to RADC. The data sources included questionnaires completed by project leaders and other project personnel, program source code listings, problem reports and system modification notices. Teams of programmers were formed to analyze problem reports and modification notices and classify the subject errors according to the TRW classification scheme. These projects have been discussed in greater detail in other parts of this report.

Proposed Data Requirements for the ROME
AIR DEVELOPMENT CENTER Software Data Repository*

- Project definition and related data
- Customer/contract definition and related data
- Subcontract identification and related data
- Software installation data
- Project organizational data
- Project employee information
- Computer equipment and support facilities identification and capability data
- Identification of project's utilization of programming methodologies, tools and techniques, and related data
- Projects programming language identification and related data
- Work package identification data with scheduled performance of tasks
- Product identification data and methodologies used in its production
- Project performance including resource expenditures data
- Project quality assurance provisions including records of all proposed changes to a baselined product configuration
  - Software Problem Report data
  - Software Modification Transmittal data
  - Milestone event data
- Information related to software operations, test case, job and total run time, and errors
- Information related to program structure, language construct usage, data structure and usage, etc.

---

*Source: Reference 11, Vol I, pp 78-87

Figure 1 Proposed Contents for the RADC Data Repository

The resulting data sets have been entered into the baseline data repository. In addition to the computerized data, narrative data and other supporting information are available.

RADC has subsequently undertaken two more data collection projects. The objective is to obtain data to assess the effects of modern programming practices on software development. In contrast to the previous data collection efforts these data are being obtained concurrently with the software development.

Program characteristics, resource utilization, and problem data are being collected by automated means as much as possible. In addition, project and environment data are being obtained with questionnaires. As back up, microfiche records are being kept of specification documents and source listings.

Automated data collection is accomplished using standard library routines available from the computer manufacturer.

One project is collecting data from a system being developed on an IBM computer system. Data such as:

- Time of run
- Lines of code added
- CPU time
- Cumulative CPU time
- etc.

are being obtained from the operating system software.

The other of the two new projects is being developed on CDC equipment. Similar data items are being obtained from a modified version of the Network Operating System (NOS).

These data will be added to the baseline repository after they are edited for completeness, accuracy, and consistency.

Errors are indicated by problem reports and modification notices. It is not possible to classify the errors until after the reported problems are corrected. Therefore, error classification is done according to the TRW taxonomy by the responsible programmer after the reported problem has been corrected.

The results of the error analysis are useful to the programmers. RADC believes that providing the programmers with this information improves the quality of the error classifications and promotes cooperation.

5.3.1.4 The Future of the DACS

Eventually, RADC expects to obtain a large collection of software data describing all aspects of software development. The data should be useful not only for RADC applications but many others as well. The Center views the DACS as a resource for all software researchers. The prototype or baseline system is presently operational on the RADC Honewell Computer equipment using a commercial data base management system. The baseline system is described in Reference 24.

RADC will soon contract for the implementation of the DACS. The initial function of the Center will be to provide software researchers with information describing the state-of-the-art in software engineering technology. As more data are added to the Center, and as enhancements are introduced into the software system, additional services will be made available. The Center is expected to be in full operation in about three years.

5.3.1.5 The Value of the RADC Repository to USACSC

In its present state the baseline repository has only limited value to USACSC. In the future when the data collection has expanded to include a range of application types, it is likely that some of the data would be useful. The first responsibility of the repository is to aid the

development of Command, Control, Communications and Intelligence (CCCI) and related software. This type of data represents a different processing environment, a different development environment, different computer systems and different languages from those used at the USACSC.

For the present these differences must limit the applicability of any results obtained from the RADC data in other environments. Certainly the methods developed for collecting data and the studies performed at RADC have implications for USACSC, but additional work must be done before it is possible to know if using the same procedures at the USACSC would lead to the same results.

The state-of-the-art in software engineering may advance to the point that some of the effects described above can be measured and procedures developed for combining data from different environments. Until then the best results can probably be obtained by RADC and USACSC sharing research findings and data in order to advance the state-of-the-art. Neither agency can afford to wait until the other solves its problems for it.

In summary, the RADC Data and Aanlysis Center for Software is an important future resource for the USACSC and the rest of the software engineering community. It can be expected to be an important source of data and knowledge about the software development process. However, it cannot substitute for a USACSC research program.

5.3.2 USA Ballistic Missile Defense System Command (BMDSCOM)

For nearly five years BMDSCOM has been collecting software data to support its software engineering program. BMDSCOM has developed and applied a large number of state-of-the-art techniques for every aspect of software development.

The Quantitative Data Base was developed to further the research programs in software engineering. Data was furnished by contractors on a regular basis. The initial collection was largely retrospective and produced poor quality data. Efforts to improve the quality of the reporting were not successful. The data base is no longer being maintained.

The Quantitative Data Base was completed from the experience developing seven software items comprising four major software systems. The five companies responsible for the software development supplied the data.

Two forms were used to report the data: The Software Development Data Form and the Software Modification Data Form. These are shown in Figures 2-a and 2-b.

The Software Development Data Form (SDDF) describes computer and personnel resources expended on various development activities. This form was submitted quarterly.

The Software Modification Data Form (SMDF) describes changes to the software items "....occasioned by the occurrence of a difficulty propogated by an earlier activity or any discrepancy which requires the expenditure or loss of one or more man-days effort to isolate, correct and test." The forms were submitted upon completion of the modification. Only one form was to be submitted for each difficulty.

The forms cover a period of about two-and-a-half years. As was stated above, some of the data describing the initial period were reconstructed by the project leaders.

The SDDFs have been compiled into reports such as the one shown in Figure 2-c. Over 150 SMDFs have been compiled into reports like the one shown in Figure 2-d.

SOFTWARE DEVELOPMENT DATA FORM

SOFTWARE ITEM NAME _____

ORGANIZATION NAME _____

REPORTING PERIOD - FROM _____ TO _____ REPORT DATE _____

ACTIVITY	MAN-DAYS	START DATE / END DATE	COMPUTER TIME		STAFF EXPERIENCE		
			7600	OTHER	< 5 YRS	5-10 YRS	> 10 YRS
REQUIREMENT SPECIFICATION							
SOFTWARE DESIGN							
CODING-DEBUG							
UNIT TEST							
INTEGRATION							
VALIDATION TEST							
ACCEPTANCE & PERFORMANCE TEST							
MANAGEMENT							

DOCUMENTATION PRODUCED	PAGES	MAN-DAYS	SOURCE STATEMENTS CODED	POL	MOL	OTHER
			TOTAL THIS PERIOD			
TEXT			LOGICAL			
MATHEMATICS			MATHEMATICAL			
DRAWINGS			CONTROL & DATA MANAGEMENT			
NUMBER OF DOCUMENTS			INPUT/OUTPUT			

REMARKS _____

Figure 2-a Quantitative Data Base Software Development Data Form

SOFTWARE MODIFICATION DATA FORM

SOFTWARE ITEM NAME _____

ORGANIZATION NAME _____

DATE _____

CURRENT ACTIVITY		MODIFICATION TYPE	
REQUIREMENT SPECIFICATION	☐	REQUIREMENT CHANGE	☐
SOFTWARE DESIGN	☐	REQUIREMENT ERROR	☐
CODING-DEBUG	☐	DESIGN CHANGE	☐
UNIT TEST	☐	DESIGN ERROR	☐
INTEGRATION	☐	CODING ERROR	☐
VALIDATION TEST	☐	DOCUMENTATION ERROR*	☐
ACCEPTANCE & PERFORMANCE TEST	☐	OTHER	☐

*TITLE OF DOCUMENT IN ERROR _____

EFFORT EXPENDED TO:	MAN-DAYS	START DATE / END DATE	COMPUTER TIME 7600	COMPUTER TIME OTHER
DETECT AND ISOLATE ERROR				
CORRECT ERROR OR INSTALL CHANGE				

SOURCE STATEMENTS CHANGED OR ADDED	POL	MOL	OTHER
LOGICAL			
MATHEMATICAL			
CONTROL & DATA MANAGEMENT			
INPUT/OUTPUT			

REMARKS _____

Figure 2-b Quantitative Data Base Software Modification Data Form

TABLE 15. ACCUMULATED RESOURCES USED - PDS-1

ACCUMULATED RESOURCES USED

SOFTWARE ITEM NAME PROCESS DESIGN SYSTEM-1 (PDS-1) THRU PERIOD ENDING 9-30-74

ACTIVITY	MAN DAYS	CALENDAR DAYS	COMPUTER TIME*		AVERAGE STAFF EXPERIENCE		
			7600	OTHER	<5 YRS	5-10 YRS	>10 YRS
REQUIREMENT SPECIFICATION	1000	832			30	30	40
____ CHANGES							
____ CORRECTIONS							
SOFTWARE DESIGN	2055	791		8000	40	30	30
3 CHANGES							
0 CORRECTIONS							
CODING-DEBUG	2623	791		22500	40	40	20
1 CHANGES							
3 CORRECTIONS							
UNIT TEST	1519	457		14000	40	40	20
____ CHANGES							
____ CORRECTIONS							
INTEGRATION	803	457		8000	40	35	25
____ CHANGES							
2 CORRECTIONS							
VALIDATION TEST							
2 CHANGES							
1 CORRECTIONS							
ACCEPT. and PERF. TEST	100	242	2000	1080	-	-	100
____ CHANGES							
____ CORRECTIONS							
MANAGEMENT	648				-	47	53
TOTAL	8748		2000	53,580	27.1	31.7	41.2

* CPU Minutes

TABLE 16. ACCUMULATED PRODUCTION DATA - PDS-1

ACCUMULATED PRODUCTION DATA

SOFTWARE ITEM NAME PROCESS DESIGN SYSTEM-1 (PDS-1) THRU PERIOD ENDING 9-30-74

DOCUMENTATION PRODUCED	PAGES	ESTIMATED MAN DAYS	SOURCE STATEMENTS CODED	POL	MOL	OTHER
TEXT	3029		LOGICAL	22070		
			CHANGES			
			CORRECTIONS			
MATHEMATICS	50		MATHEMATICAL	4414		
			CHANGES			
			CORRECTIONS			
DRAWINGS	10		CONTROL-DATA MGMT.	17656		
			CHANGES			
			CORRECTIONS			
			INPUT/OUTPUT	8828		
			CHANGES			
			CORRECTIONS			
TOTAL	3089	1252				
			TOTAL	52,968		
NUMBER OF DOCUMENTS	12					

Figure 2-c Sample Summaries of Resources and Production from the Quantitative Data Base

TABLE 31. SUMMARY OF MODIFICATIONS - BL/CISS SIM

SUMMARY OF MODIFICATIONS

Software Item Name COMPUTER-INDEPENDENT SYSTEM SPECIFICATION (CISS) SIMULATION SYSTEM

Date	Days Outstanding	Computer Time And Activity of Detection							Number of Statements And Modification Type							Man Days		
		RS	SD	CD	UT	IT	VT	AP	RC	RE	DC	DE	CE	DD	OT	DI	CI	TOTAL
9-1-73		X							12	102					6			335
									2	145	34	38	189	21				225
to						X			1	37	17	27	111	13	2			380
6-30-74							X			16	6	2	20					50
5-8-74	24					40				32						5	10	15
5-6-74	137					182				88						15	22	37
5-24-74	1					15			85							0	1	1
5-24-74	18					72				4						5	10	15
6-25-74	4					33				38						.5	2	2.5
6-10-74	12					73						30				2	5	7
6-10-74	9					28				9						2	4	6
6-3-74	18					71				10						1	8	9
6-4-74	15					25				3						3	2	5
6-19-74	10					36				12						4	2	6
7-16-74	7						24			6						1	2	3
7-8-74	10					30				5						2	2	4
7-24-74	24					72				12						10	5	15
7-1-74	177						340			271						5	50	55
7-8-74	36					296				170						5	20	25
7-1-74	30						500								185	20	10	30

Figure 2-d ample Summary of Modifications from the Quantitative Data Base

5.3.3 USAF Data Systems Design Center (DSDC)

The DSDC is the Air Force's counterpart to the USACSC. Standard software items are developed and maintained for more than 140 installations worldwide.

The DSDC has developed an excellent procedure for designing, coding, testing, and maintaining their software.[20] At the heart of the procedure is the Planning and Resource Management Information System (PARMIS).[21]

PARMIS is designed to facilitate the management of software development projects. It is an online system of project-related data which is updated on a weekly schedule. PARMIS has a history file representing over 2000 projects covering a seven year period.

Figure 3-a is a summary of the items available on the PARMIS history file. The file is maintained on magnetic tape and has been used by GRC[10] and Gehring[22] to study the software development process.

In addition to the project data recorded in PARMIS, DSDC maintains a semi-automated problem reporting and disposition file. The basic form is the problem report. Its purpose is to ensure that problems experienced in the field are responded to quickly and satisfactorily. The reports are kept on file for one year. Abstracts of each problem report are kept on an online file using the ARPANET computer network. Figure 3-b lists the items included in the data file.

SUMMARY OF HISTORY-FILE DATA ITEMS

(EACH ACTIVITY)

1. Project originator number
2. Activity group number
3. Control number
4. Activity number
5. Activity description
6. ADP system number
7. ADS number
8. Management category
9. Milestone number
10. Type of computer
11. Work category
12. Data systems designator number
13. System code
14. Type of system
15. Program action code
16. Program number
17. Schedule indicator
18. Responsible individual for data
19. Privacy key
20. All predecessor activities (control and activity numbers)
21. All successor activities (control and activity numbers)
22. Activity description
23. Start date
24. New start date
25. Estimated completion date
26. New estimated completion date
27. Actual completion date
28. Plan change date
29. Span days
30. New span days
31. Remaining span days
32. Estimated man-hours by skill and total
33. New estimated man-hours by skill and total
34. Expended man-hours by skill and total
35. Monthly expended man-hours by skill and total
36. Current expended man-hours (since plan change date) by skill and total

Figure 3-a Summary of Items from the PARMIS History File

Items in DIREP File

DIREP Number

System ID

Date Problem Reported

Date Report Logged In

Office of Primary Responsibility

Date OPR Response

Problem Description

Severity

Correction Date

Release Date

Figure 3-b List of Items in the AF Data Systems Design Center Problem Report File

6.0 CONCLUSIONS

The evidence obtained from an analysis of available literature together with conversations with researchers indicates that there is only a very limited possibility of using existing data from other sources at USACSC. This conclusion is based on the following facts:

- Investigators using existing data have reported internal inconsistencies and have generally criticized its quality. This gives little encouragement to use such data at the USACSC thereby further removing it from its origin and the persons who developed it.

- All data sets of any size were collected after the fact. Reporting cannot be complete or accurate when the data are reconstructed from available documentation sometimes without the participation of the originators of the data.

- Classification criteria were applied inconsistently. Data users criticize the classification schemes and the way they were applied. They report instances where similar results may have been classified differently.

- Users complained that no documentation is available describing the methods for applying the classification criteria for error and other data.

- Some error data exists that may be of acceptable quality, but it was collected in an environment different from USACSC and representing different types of applications. Detailed supporting information is not available that describes the programs, design, development methods, test procedures, etc. This would be helpful in gaining some insight into the differences that might affect the interpretation

of the results and their implications for the USACSC. However, even if the information explaining these differences were available, there are no reliable ways of correcting the data for use at the USACSC.

- The statements of researchers repeatedly caution about using the data in different environments or indicating the need for contact with the originators for interpretation.

It is possible that we have not located every source of error data. However, there is no reason to expect that data that was of significant value to some researchers has escaped presentation in the literature included in the survey.

The results obtained using the data collected under controlled conditions have not been good. Using the data in another environment would simply add more unknown effects to confound any analysis. Furthermore, using it to supplement USACSC data introduces the possiblity of corrupting the USACSC data. In almost all cases the quality of the data, especially the circumstances under which it was collected is unknown.

In summary, there are reasons to have serious doubts about the quality of available software data. Using the data as part of USACSC's research program would jeoparidze the outcome from the start. It would not be possible to know if failures to develop relationships were the result of wrong hypotheses or bad data.

From the beginning, software engineers have decried the quality of available data. It seems self-defeating to undertake a research program without giving the utmost consideration to data quality. From this viewpoint the existing data are not useable at USACSC except for establishing directions for initial hypotheses in the absence of any other indications. If existing data indicates a result that conflicts with intuition, it would probably be better to trust intuition.

7.0 RECOMMENDATIONS

In recommending the research program described below, our objective is to do more than develop a few procedures or techniques that improve program quality. We propose a systematic analysis of all aspects of the software development process. We want to develop basic principles that can be demonstrated to be cost effective. It is not enough to observe that a certain coding practice seems to produce code that is less error prone. We want to know if the effect is repeatable and under what conditions. We want to know the cost of the process in terms of programmer, management, and computer resources. Furthermore, we would prefer an integrated set of methods to isolated techniques.

Our recommendations are based on the following conclusions:

- The data collected to date is not satisfactory for USACSC use.

- The findings of other researchers regarding the causes of errors cannot be applied to the USACSC environment with any quantifiable expectation. There are no definitive measures available that describe the cost-effectiveness of various schemes for improving program quality. This includes:

  - structured coding
  - chief programmer teams
  - testing methods

  We are not suggesting that these methods do not produce the desired results. But, we believe that the results to be achieved are not predictable. There is not sufficient evidence to determine if any advantage will be achieved in a given environment with given programs.

- No methods have been developed and demonstrated that can predict the error content of a program with any acceptable confidence.

Our recommendations are directed toward the achievement of the following objectives:

- Developing quantitative descriptions of the errors most often encountered in released USACSC programs.
- Developing an understanding of why these errors occur.
- Preparing cost-effective ways of avoiding the errors.
- Preparing cost-effective ways of eliminating the errors prior to release.

Considering the state-of-the-art in error data collection and analysis and the objectives, we make the following recommendations.

1. Preliminary error data collection

Our analysis of previous efforts at error data collection indicates that classification schemes tend to become unmanageable if some limits are not imposed. However, it is not possible, a priori, to construct good limits. An iterative process is required.

Previous efforts have also demonstrated that good error classification systems cannot be created in a vacuum. Schemes that appear to have merit from a purely structural viewpoint may prove to be difficult to implement. Also, the construction of the categories should reflect the frequencies of occurrence of the errors. There should not be large numbers of classes that represent few or no errors.

Therefore, we recommend that a preliminary period of error data collection be undertaken by the USACSC as a first step in the development of a practical error classification system. The collection should be performed for a relatively short period of time (say 3 to 6 months) and be presented to everyone as experimental.

During the collection period different methods of capturing data should be tested. The experiences classifying the errors should be used to improve the definitions of the classifications. The following results should be obtained:

- Frequency distributions of errors by class
- Descriptions of problems with data collection
- An estimate of the cost of data collection in terms of time and funds
- Improved definitions
- The cooperation of the affected parties
- Some indication of the principal causes of the errors

Use of the data collected during the initial effort should be approached with caution. It would tend to suffer from the same deficiencies that affect existing software data. Therefore, it will be necessary to review and reclassify the data after the final procedures are developed. Any data whose quality is suspect should be discarded. After all, the primary purpose of the initial data collection activity is to develop definitions and procedures that will improve the quality of subsequently collected data.

2. Error classification system

The results of the preliminary error collection effort should be used to develop a formal classification system. Formal definitions of variables that may contribute to the introduction of errors such as program size, complexity, etc., and project variables must be prepared.

The classification scheme should have the following characteristics:

- An overall conceptual framework that is:
  - Compact. Five or at the most ten major categories.
  - Language and machine independent.

- Exhaustive. It should cover all types of errors without necessitating the addition of special categories for unusual cases.
- Exclusive. An error should only be assignable to one category.
- Specific. The definitions of each class should leave no doubt about what errors are included

- A succession of subcategories that can become specific enough to distinguish among error types. This may require language and machine related definitions.
- Able to distinguish between errors and problems.
- Precise. It must be possible to make any necessary fine distinctions between errors that may appear similar.
- Reliable. Different persons using the system in different locations should arrive at the same classifications.
- Valid. The classifications should describe real errors and not attributes of programs or improvements.
- Documented. Complete, written definitions and procedures are essential.
- Portable. It must be applicable in different environments.

Preferably the major categories would be very stable. They should be defined in a way that would not require changing. It would be idealistic to expect that the initially developed definitions would never be changed, but any subsequent proposed changes should be scrutinized very carefully. The existence of a well-constructed superstructure would mean that any subsequent research could be accommodated by the classification system and would maintain a basis for making longitudinal comparisons. It would mean an ever expanding data base with consistent definitions at some useful level of detail.

It must be recognized that certain problems and their related errors are machine and language dependent. These considerations and the need for discrimination among similar errors can be accommodated in the subcategories of the classification scheme. Here we must also deal with the problems of assignment to exclusive classes and distinguishable subcategories. We also must consider the implications of future changes in equipment and supporting software.

One possible starting point for the construction of the error classification system is the method developed by MITRE Corp. It uses a a logical hierarchy which includes both errors and problems. It is machine and language independent at the highest level. Its major shortcomings may be its large numbers of categories and its acceptance of overlapping error subcategories.

The RADC system must be considered carefully. It may be possible to improve on the system and obtain a system that possesses the attributes described above. By proper definitions of the USACSC categories it may be possible to make better use of the RADC data repository.

3. Development of hypotheses.

It is difficult to conceive of a successful error data collection program that does not have a clearly defined purpose. Unless we know why we want the data and what we are going to do with it, it is difficult to limit and control the effort. Data requirements become vague. We collect everything imaginable in order to cover all possibilities. We have no basis for establishing accuracy requirements. We don't know how to budget limited resources. It becomes difficult to enlist the support of management and project leaders if they fail to see any direct benefits from their efforts.

Past experience has shown that it is better to collect limited data of good quality than to try to capture great amounts of data whose quality is open to question or is unknown.

Some possible hypotheses to be tested include:

- The number of errors reported during the first 6 months of system operation is a linear function of lines of source code.

- The number of errors reported during the first 6 months of system operation is a linear function of other characteristics of the code which describe its complexity.

- The use of structured coding reduces the number of errors reported during the initial operation of the system a significant amount.

- A given model is a reliable predictor of the number of errors reported during the first 6 months of operation.

- The primary causes of errors detected during testing and integration are:
  - Failure to fully specify requirements
  - Improper interfacing between program modules
  - Bad logic paths
  - Values exceeding specified limits

Many other hypotheses can be prepared. Each will help to define specific data requirements. When used with the previously developed error classification scheme, the data needs can be defined with precision. Methods for collection will be developed that are consistent with the necessary accuracy.

The testing of hypotheses that attribute errors to selected program characteristics, development methods or coding structures is preferred over statistical procedures involving the systematic manipulations of large numbers of variables. Hypothesis testing makes better use of informed judgement regarding the factors that contribute to errors. It also reduces the number of trails required to obtain statistically significant results.

4. Develop data needs.

When we use the method of hypothesis testing for the construction of our experiments, the development of data needs becomes much simpler. The hypotheses serve to define the variables to be collected. They also give a basis for specifying accuracy. It also serves to define the set of items from which data will be collected. If we are trying to find the difference between two procedures, such as in determining the effect of structured coding practices, we would want all other factors to be as controlled as possible. The programs under consideration should be of a similar type and complexity. The hardware/software should be the same. The development environment should be controlled. Any sources of data failing to meet these requirements should be rejected.

At times the development of data needs may affect the order of the studies. If some of the variables prove impractical to control, it may be necessary to schedule studies to quantify the effects of the uncontrolled variables.

The data needs definition should include the following:

- A list of the required data items
- Number of measurements for each item
- Required precision for each item
- Timing compatibility among the items. (e.g. the program size measurement must be consistent with reported errors. This requires the size measurement to be correlated with the time of the error identification)
- Priorities
- A list of the variables to be controlled
  - e.g. - program size
    - program complexity
    - method of coding control
    - average experience of programmers
    - application type

5. Data collection.

The most important step in the entire process of defining data needs and collecting the data is to develop a procedure for collecting the data so that when the results are analyzed, it is possible to know exactly what was collected (definitions), how it was collected, and any factors that may influence the interpretation of the data.

To accomplish this requires the explicit understanding among all participants that the proper execution of this process requires time and resources. It also needs the support of both management and the authors of the programs.

A well-designed data collection system must include checks for quality and compliance. It must not overly tax the resources required to complete the software development. And finally, it must not alter the process or otherwise influence the outcome of what it is trying to measure. This last restriction is an ideal which can never be entirely achieved.

The data collection process must include:

- Designed and tested data collection instruments
- Written definitions for all variables
- Written procedures for data measurement and processing
- Management awareness and support
- A method for reporting and solving problems

6. Analysis.

The findings and results of the data collection should be reported formally to all participants. A mechanism should be developed for obtaining comments and suggestions for improving the data collection and for future studies.

The ultimate objective of the analysis is to quantify the findings and to establish a basis for future studies.

8.0 REFERENCES

1. Grooms, David W., Computer Software Reliability (a Bibliography with Abstracts), NTIS, Springfield, Va., July 1977, NTIS/PS-77/0572

2. Amory, W., Clapp, J. A., Engineering of Quality Software Systems (A Software Error Classification Methodology), MITRE Corp., MTR-2648, Vol VII, Jan 1975, also RADC-TR-74-325, Vol VII

3. Thayer, T. A. et al, Software Reliability Study, TRW Defense and Space Systems Group, RADC-TR-76-238, Aug 1976

4. Farr, L., Zagorski, H., Factors That Affect the Cost of Computer Programming; A Quantitative Analysis, System Development Corp., TM-1447/001/00, Aug 1964

5. Weinwurm, G. F., Zagorski, H., Research into the Management of Computer Programming; A Transitional Analysis of Cost Estimating Techniques, ESD-TR-65-575, Nov. 1965

6. Nelson, E. A., Management Handbook for the Estimation of Computer Programming Costs, System Development Corporation, RM-3225/000/01, 20 Mar 1967

7. LaBolle, V., Development of Equations for Estimating the Costs of Computer Program Production, System Development Corporation, TM-2918/000/00, 1966

8. Nelson, E. A. Fleishman, T., A System for Collecting and Reporting Costs in Computer Program Development, System Development Corporation, TM-3411/000/00 11 Apr 1967

9. - Automatic Data Processing Resource Estimating Procedures (ADPREP), Planning Research Corporation, PRC R-1527, Aug 1970

10. Graver, C. A., et al, Cost Reporting Elements and Activity Cost Tradeoffs for Defense System Software, General Research Corporation CR-1-721, Mar 1977

11. Willmorth, N. E., et al, Software Data Collection Study (8 Vols), System Development Corporation, TM-5524/001/01, Dec. 1976

12. Shooman, M. L., Bolsky, M. T., Types, Distribution and Test and Correction Times for Programming Errors, Proceedings, 1975 International Conference on Reliable Software, Los Angeles, April 21-23, 1975

13. Endres, Albert, An Analysis of Errors and Their Causes in System Programs, IEEE Transactions on Software Engineering, Vol SE-1, No. 2, June 1975, PP 140-149

14. Baker, W. F., Software Data Collection and Analysis: A Real-Time System Project History, IBM Corporation, RADC-TR-77-192, June 1977

15. Willman, H.E., Jr., et al, Software Systems Reliability: A Raytheon Project History, Raytheon Company, RADC-TR-77-188, June 1977

16. Motley, R. W. Brooks, W. D., Statistical Prediction of Programming Errors, IBM Corporation, RADC -TR-77-175, May 1977

17. Bakkegard, I. G., Quantitative Data Base, System Development Corporation, TM-HV-195/000/00, Apr 1975

18. Hansen, Gregory A., Measuring Software Reliability, Mini-Micro Systems, Aug 1977, PP 54-57

19. Sukert, Alan N., A Software Reliability Modeling Study, USAF, Rome Air Development Center, RADC-TR-76-247, Aug 1976

20. - The AFDSDC ADS Project Approval/Development Process, AFDSDCM 300-8 (TEST), AF Data Systems Design Center, Gunter AFS, June 1976

21. - Planning and Resource Management Information System, AFDSDCM 300-405, AF Data Systems Design Center, Gunter AFS, Sept 1976

22. Gehring, P. A., A Quantitative Analysis of Estimating Accuracy in Software Development, Texas A&M University, Aug 1976

23. Boehm, Barry W., Software and Its Impact: A Quantitative Assessment, Datamation, May, 1973, pp 48-59

24. Duvall, Lorraine M., Software Data Repository Study, Illinois Institute of Technology Research Institute, RADC-TR-76-387, Dec 1976.

APPENDIX A

QUESTIONNAIRE AND CHECKLIST

CONTACT NO. _____ DATE _____

SOURCE OF REFERRAL:

INDIVIDUAL CONTACTED:

TITLE:

AFFILIATION:

ADDRESS:

TELEPHONE:

1. Has your organization been engaged within the past 5 years in studies related to software errors, quality or reliability?

a.	Title			
	Sponsor	Project Officer	Location	Telephone
	Project Leader	Present Affiliation	Location	Telephone
b.				
c.				
d.				
e.				

2. Were data describing actual software errors used in any of these studies and if so were they
   a. Collected for the study, __, __, __, __, __
   b. Existing __, __, __, __, __

3. Where is the data presently located?

4. Is it available to government agencies?

5. Who is to be contacted regarding access?

Name	Title	Affiliation	Location	Telephone

6. Have other researchers used the data?

	Name	Title	Affiliation	Location	Telephone
a.					
b.					
c.					

7. What publications have described the data base or studies that were made using it?

	Title	Author	Affiliation	Date
a.				
b.				
c.				

8. Do you know of other software error data bases?

	Organization	Contact	Location	Telephone
a.				
b.				
c.				

9. Who are other individuals or organizations doing research in software errors, quality or reliability?

	Individual	Affiliation	Location	Telephone
a.				
b.				
c.				
d.				
e.				

10. What other organizations are sponsoring studies in software errors, quality or reliability?

	Organization	Location	Contact	Telephone
a.				
b.				
c.				

. Detailed Description of Software Error Data Base

General Description of Data Base

1. Purpose of the data base.

2. Period of time represented by data entries (not necessarily same as period of data collection).

3. Participating organization.

4. Storage medium

5. Hardware and software used to create and maintain the data base.

6. Files contained in the data base.

7. Approximate numbers of records in each file.

8. The records are: complete__,Mostly complete__, half complete__, less than half__.

9. Is a distinction made between a problem such as a user might report and the actual cause of the problem or error as a programmer might describe it?

10. Are the problem and error categories described in a formal document? How can we obtain a copy?

11. Are sample copies available of the problem and correction reports?

12. Are these the source documents for the data base?

13. Other sources used.

14. Is an error report required for each problem or can a developer release a correction without an error report.

15. Are enhancements or changes separated from program errors?

16. How are problems and errors related to each other in the data base?

17. Can more than one problem be tied to a given error.

18. Can an error sequence be tied to a single problem.

19. Can the module or program identified with the problem be separated from the ones with the errors?

20. Is a narrative text included as part of the computer record?

Method of Error Data Collection

21. Does a person other than the developer or user verify and categorize the problem?

22. Does a person other than the developer or software maintenance person verify and categorize the error?

23. How are duplicate problem reports eliminated?

24. Can the same problem be recorded differently because it is reported by different users?

25. Are the problem and error definitions clear and distinct?

26. How is the development data obatined?

27. Are development hours recorded by activity or by total project phase?

28. Were the data collected during development and test or reconstructed afterward?

29. Have there been changes in definitions during the lifetime of the data base?

CHECKLIST

Error Data

- Program or module ID
- Date problem discovered
- Problem classification
- Who discovered problem
- Who verified problem
- Error classification
- Who classified error
- Who verified classification
- Affected modules
- Date corrected
- Manhours to correct
- Computer hours to correct
- Net change in size of program

Development Data

- Manufacturer of computer hardware
- Configuration
- Operating software
- Language
- Number of modules or programs
- Number of lines of source code per module
- Number of lines of object code per module
- Software type e.g. Real time, interactive, batch
- Application
- Developing project organization
- Method of development control
- Programming management or control
- Space constraint
- Time constraint
- Amount of code previously existing
- Amount of library or standard code

- Analysis hours
- Design hours
- Programming hours
- Testing hours
- Operational date
- Original scheduled Operational date
- V&V organization
- V&V software or equipment
- V&V procedure
- Number of design changes
- Timing of changes
- Hours of field testing
- Original project manhour estimate
- Was test specification part of system specification
- Development inhouse or contracted
- Type of contract
- Dollar amount
- Pages of functional and detailed specs

Operational Data

- Number of installations
- Operational configuration
- Total number of hours per month
- Reporting cycles
- Total number of users
- Mode of operation
- Total hardware downtime per month
- Hours of software maintenance per month
- Number of maintenance sites

APPENDIX B

SUMMARIES OF STUDIES

Title: Software Reliability Study

Topic: Software Reliability

Objectives:

Define software data requirements and present methods for collection by compiling and analyzing data from four software development projects:

Complete a survey of software reliability models and present Nelson's theory of software reliability.

Authors: G. R. Craig, L. E. Frey, W. L. Hetrick, M. Lippo, T. A. Thayer, J. A. Yoxtheimer, J. A. Whited, R. B. White

Affiliation:

TRW Defense and Space Systems Group

Sponsor:

USAF, Rome Air Development Center (ISIS)

James V. Cellini, Jr. Project Engineer

Discussion:

The two objectives of the study are not closely tied together in the report. Little effort was made to reflect the needs of the theoretical models into the data collection requirements or to validate the theories using actual data. Therefore, since our main interest is error data collection, the following descriptions of the project and the results is limited to the first objective, which constituted the major portion of the effort.

The starting point for the project was the data collected from four large software development projects. Neither the names of the projects nor their developers are described in the report. This is standard practice for RADC-sponsored software research projects. The only descriptions of the type of software on which the collection methods and analysis techniques were developed is that there were two large command

and control systems, one data management system and one real-time, highly analytical system.*

The data source documents for three of the systems were problem reports and change notices that were designed for configuration management. Classification of errors was done after the fact by the persons responsible for the development, if possible.

A classification method is presented which is a modified and scaled down version of a scheme which has been developed and modified by TRW over a series of previous studies. The report states that the MITRE classification scheme was used, but it is not clear how this could be. The present scheme has 19 major categories. It is highly language dependent and contains ambiguities and duplications.

The data for the fourth project (real-time system) was designed and collected concurrently with the software development. This collection was not conducted using the methods presented in the report. They were prepared using the experiences described in the report.

The error data was used to test the relationships between the incidence of errors and various software and project characteristics. However, because of the limited data describing the software and project characteristics, very little analysis could be done. The report devoted space to how the characteristics could be obtained in the future. Several automated software analysis tools were described.

Some statistical analyses were performed using modules of some of the projects for which characteristics were available. The variables considered included the number of source statements and certain measures believed to be related to program complexity.

---

*The report warns about applying the data or the results to other types of system or development environment.

Five components of complexity were presented along with measurable quantities that can be evaluated by an automated tool called 'TMETRIC. The use of the tool is limited to the JOVIAL J4 language, as are the measures of complexity. Several modules were studied using standard statistical techniques to relate the complexity components to error production.

The error data were measured during four testing periods: development, validation and acceptance, integration, and operational demonstration. Data were not uniformly available for all projects.

Results:

A software error classification system is presented which has 19 major categories and 162 subcategories. The classification includes symptoms and causes, but these groups are not explicitly separated. The report states that it was not always possible to determine the causes of failures when classifying errors. Also that time and accuracy are lost if error classifications are done either after the fact or by persons other than the author of the program.

An analysis was made of when errors were found. An attempt was made to relate these to when they were introduced.

Analyses were made of how errors are related to

1. module size
2. module or routine complexity
3. development method

The results although promising and consistent with the findings from other studies could not be made on all the projects because of inconsistencies in the data sets. The findings were not believed to be generally applicable. Also, as was mentioned above, some of the data were not reliable because of the delay in their collection. Furthermore, measures of the software characteristics were not necessarily reflective of the modules that produced the errors. That is, the software is

dynamic during development and many errors may be introduced by software configurations that are different from when they were analyzed using the language monitor.

Some general guidelines for collecting error data were presented

1. Problem reports and closure reports should be problem oriented. They should include the symptoms of the problem, an accurate statement of the problem, and a detailed description of the necessary fix.
2. Data collection and analysis should be preceded by standards specifying procedures and formats.
3. All project performers should be made aware of the objectives of the study.
4. Further work to generate reliability data collection tools is necessary
6. The data collection must be started early in the software development cycle.

Data Used:

Problem reports and modification notices from three projects. The forms were designed and used for configuration management. The problem and modification forms for the fourth project were available as a single document. In all cases the classification of the errors and their causes were recorded after the fact by the program author if possible otherwise it was done by a knowledgeable programmer.

Some source programs for which error data was available were run through an automated language analyzer which computed certain program characteristics.

Source:

The source of the data was not disclosed.

Title: A Software Reliability Modeling Study

Topic: Reliability Model Validation

Objectives:

Compare predictions of nine reliability models with actual data to determine if they work well for projects other than those used to validate them originally.

Author: Captain Alan N. Sukert

Affiliation: USAF, Rome Air Development Center (ISIS)

Sponsor: Same

Discussion:

Rome Air Development Center (RADC) has as a major objective the investigation of factors that contribute to the development of reliable software for command and control and intelligence systems. As part of this objective RADC has let a series of contracts for developing methods for measuring and predicting software reliability, classifying software characteristics and errors, and for collecting relevant data.

In this report nine reliability models including some commissioned by RADC as well as others published in the literature were exercised using data collected from a large command and control system development. The system and the developer were not identified, but the characteristics were the same as one of the systems presented in the TRW Software Reliability Study.

The data were extracted from System Problem Reports and Software Modification Notices which were initially designed for configuration control. The errors were classified by either the authors of the programs or knowledgeable individuals as part of a previous study. Using the error classification and other information available either from the previous study or from the SPR/SMN the computerized data was sorted to remove non-software errors and to place them into chronological order.

The date of the error was taken from the SPR which placed it into one of four test phases; Validation, Acceptance, Integration and Operational Demonstration. The included time period was about five months.

Some inconsistencies in the entries describing the errors and their disposition led to the creation of four data sets covering all possible interpretations. (Attempts at reconciling the inconsistencies suggested that there may have been some problems in implementing the original data reduction procedures.)

Some difficulties occurred in establishing a suitable time base. Furthermore, all the modules were not in test at the beginning of the test period. Finally, some of the models required computer CPU time as opposed to the available calendar time. An attempt was made to execute these models using the available time base.

The models were executed using two different origins for the time measurement and the day and the week as the time increment.

Reliability predictions were made using the four data sets, the two starting points, and two time increments.

Results:

The measures predicted differed among the models. The models were basically analogs of hardware reliability models featuring mean time to failure (MTTF), numbers of errors remaining and probability of failure in the next time interval.

The results varied widely making it impossible to vigorously establish the validity of any of the models. Some tendencies were detected by the author which should form the basis for future studies.

Source:

The exact source of the data was not disclosed.

Title: Structure and Error Detection in Computer Software NPS555s75021, Feb. 1975 (AD A014 334)

Topic: Effect of Program Structural Complexity on Error Detection

Authors: Bradley, Gordon H., Green, Thomas, Howard, Gilbert T., Schneidewind, Norman F.,

Sponsor: Naval Air Development Center, Warminister, PA

Contract: N62269/75/RO/02014

Author Affiliation:

Bradley, Howard, Schneidewind — Prof. Dept OR & Admin. Sci., Naval Post grad School Monterey, Calif.

Green, LT, USN

Complex programs are more difficult to test than simple ones. It is desirable to find the relationship between program structure and error detection and test. A simulation model has been prepared to test this relationship.

Structure can be controlled throughout the development process. Except in cases of extreme performance requirements it should be possible to keep program structure simple. Want to be able to control costs by controlling structure. There is substantial evidence that shows the large portion of total development cost is incurred during testing. This is the object of control.

Debugging is the detailed tracing of paths through a program to find errors. Testing is the selection of input data to produce a result which is compared with the desired result. The former requires a detailed knowledge of the program structure, the latter does not. Programs with a great many inputs cannot be tested adequately because of the great number of input combinations.

Testing is a critical part of the development process, here is how we can tell if the product does what it is supposed to. It eats up a large portion of the development resources. It indicates how

reliably the product will perform in the field. We want to investigate how error detection during testing is affected by the structure of the program. Program structure means how the different parts relate to each other.

A program is modeled as a set of arcs representing flows of instructions between nodes which are program branch points. Each path is dictated by the input data and may contain an error. The model follows a path until an error is detected. It is corrected with some probability of introducing a new error. The path is begun again and the process is repeated until the end is reached without an error.

Two major assumptions in the construction of the model are:

1. The structure of the program is not well understood
2. No learning takes place during the encounter of errors

Measures of error detection are:

1. Number of errors detected in a fixed time
2. Number of errors detected for a fixed number of inputs
3. Pct. arcs traversed by 1 or more inputs
4. Mean time between errors
5. Pct errors remaining

Measures of program structure:

1. Number of nodes
2. Ratio of actual number of arcs to max no. if only allow one arc between nodes
3. No. of paths
4. Avg no. of arcs/path

Use the model to analyze the effect of complexity on error detection in the design phase. Use the detailed design flowchart to construct model, model is seeded with errors, then select inputs at random. Exercise the model and measure the error detection characteristics

using different levels of complexity. Do a statistical analysis to relate the complexity to the measure of error detection. Error detection characteristics are good measures of testing resource requirements.

An example is given using a hypothetical path network. The method of selecting paths from random inputs and counting the resulting errors is discussed (p13). The introduction of new errors is described.

The simulation is run using several programs of differing complexity. No conclusions were drawn.

Title: Classification Methodology, MTR-2648, Vol. VII, January 1975 (AD-A007 772)

Topic: Method for Classifying Software Errors

Objective:

To develop a method of classifying SW errors so that relationships between errors and their cause can be established. Purpose is to avoid errors or develop better ways of detecting and eliminating them in order to reduce the numbers of residual errors in delivered SW. Also make the development process more efficient by reducing effort spent locating and correcting errors.

Authors: Amory, W., Clapp, J. A.

Sponsor: RADC (ISIS) P.O. Richard A. Robinson 315-330-7546

Contract: F19628-C-73-0001

Author Affiliation: MITRE Corp.
P. O. Box 208
Bedford, Mass. 01730

RADC technical direction responsibility for task:
SW Sciences Sect (R. Nelson, Chief)
Information Processing Branch (F. Tomaini, Ch)
Information Sciences Div., RADC

Part VII of an VIII part task in SW reliability and timeliness.

Discussion:

The classification method is based on applications experience of the authors.

It is open ended and intended to be modified by results of applying it to data collection experience.

Neither the inclusion of related classification work nor the classification of live data was attempted. It was held that the experience of actually attempting a classification scheme would put both these efforts on firmer ground.

The classification scheme exercise was aided by a previous attempt to collect error data from a real time text processor and also from a literature search. Neither effort uncovered an existing class scheme of the scope described herein. The problem lies in not having data to confirm and extend the classification method.

The programmer is central to the SW development and error generating process. His interaction with the computer and development process are also important to the understanding of errors.

An hierarchal classification scheme is presented based on the following 5 basic dimensions:

1. Where did the error take place
2. What did the error look like
3. How was the error made
4. When did the error occur
5. Why did the error occur

The scheme is intended to permit a researcher to pick the categories and subcategories that are relevant to his experiments, although some reservations are stated regarding the possibility of picking the right set ahead of time.

Problem of classifying an error in more than one category (p23) is addressed saying that it might not be possible to limit each error to one category. Discusses theory of "fuzzy sets" as way of distributing errors among categories.

Title: Software Systems Reliability: A Raytheon Project History, RADC-TR-77-188, June 1977

Topic: Collect and Analyze Software Error Data

Authors: Willman, Jr., H. E., Beaureguard, A. A., James, T. A., Hillcoff, P.

Sponsor: RADC PO: James V. Cellini, Jr., (ISIS)

Contract: F30602-76-C-0140

Author Affiliation: Raytheon Co., Bedford Labs

Discussion:

Task provided a SW error data base to support further research in SW error analysis and SW error prediction model analysis.

Complete error history on SW for a large DoD ground-based radar-data-processing-dominated system.

Error data base extracted from 2165 Software Problem Reports (SPRs) written against 109 operational SW modules

3 Files:

1) Module Description File (109 entries)
2) Software Prob Rept File (2165 entries)
3) Error Category File (193 entries)

Fault taxonomy is based on a modification of TRW (Thayer et al Software Reliability Study, RADC-TR-74-250, Oct. 74)

Programs or modules are developed individually from specs and unit tested. Release notice is generated. Build is assembled from modules and integration testing takes place. 1984 out of 2165 SPRs occurred at this time (92%). Build is then released for acceptance testing. 19 SPRs generated at acceptance testing. During operational phase an additional 162 SPRs were submitted.

SPRs may be generated by anyone including the program author. They are prepared as soon as a problem is identified and are not delayed until a solution is devised and tested.

SPRs must result in a properly approved Software Modification Notice (SMN) even if error is not legitimate.

However, SMNs can be prepared without an SPR (822 of these occurred).

The control activity maintains a log of SPR/SMNs.

The subject system uses a multiprocessor arrangement with a common data base designated as COMPOOL. The majority of the software is written in JOVIAL/J3.

The system was constructed by a combination of top-down and bottom-up techniques. Development and integration of the system was accomplished through a succession of "builds" which represented increments of hardware and software capability. The purpose of a particular build was to check both the interrelationships among the software modules and the program interfaces with the new hardware (some of which was being tested for the first time under realistic conditions).

Program modules not part of a given build were replaced with dummy modules. Driver programs performed whatever functions were necessary to make the system work.

Unit testing was done on an 1108 simulator rather than the object computer. This generated some I/O SPRs.

A test plan was conceived for each program module as it was being developed.

When design of a module was completed, a detailed Test Procedure was prepared which included steps, data for input and output.

All build testing took place on a real machine.

Unit testing was completed by program authors. The program was then released to integration team.

The more advanced builds were released as integrated hardware software packages for testing in the field.

The Applications Software Dept at Bedford Labs has collected a file of approximately 10,000 SPRs/MNs. This central file was source for the 2065 SPRs/MN related to the subject software development.

Should be emphasized (see p 6-6) that not all discovered errors had SPRs. 38% of the SMNs were released by the programmers without accompanying SPRs.

There is no way to know how many modules may have been affected by an SPR or even if the module with which the SPR was associated was actually the one in which the error occurred.

Title: An Analysis of Errors and Their Causes in System Programs, IEEE Trans in SW Eng. Vol. SE-1 No. 2 June 1975

Author: Endres, Albert

Sponsor: Internal

Contract: NA

Author Affiliation: IBM Laboratories, Boebilingen, Germany on leave at Univ. of Stuttgart, Stuttgart, Germany

Discussion:

All irregularities found during testing (or suspected) were documented according to their external manifestation - the effect they produced in a specific test case. This information is called the problem.

Problem was passed to original development group which analyzed prob. and prepared error protocol which classified problems into following groups.

Problem Groups

1. Machine error
2. User or operator error
3. Suggestion for improvement
4. Duplicate of a previously defined prog. error
5. Documentation error
6. Program error (not prev. identified)

Data base contained 740 problems of which 432 were group 6: programming error

Each error protocol contained following information.

1. Administrative data on discovery of the problem.
   (system version, config, test case, date of test, name of tester, etc.)
2. Description of the problem
3. Admin data on correction made
   (changed modules, date of change, name of programmer, system version into which correction to be integrated, etc.)
4. Code for error cause, originating ... project
5. Description of the correction made.

Typical Problem Descriptions

1. System caught in loop
2. Device X could not be started; data set Y could not be read
3. System stopped with an invalid operation code; invalid storage, disk, or device address.
4. Card reader K runs with only 1/2 its theoretical speed.

Number of errors is equal to number of problems. (∴ if in correcting problem discovered and corrected other problems there won't be counted)

Error classifications

A. Failure to understand prob or way to solve it (46%)
B. Error in implementation of process (38%)
C. Other than programming errors (16%).

Causes of errors

1. Technological
   (definability of the problem, feasibility of solving it, available procedures and tools)
2. Organizational
   (division of work load, availability of information, communication, resources)
3. Historic
   (history of the project or the program, special situations, external influences)
4. Group Dynamic
   (willingness to cooperate, distribution of roles inside the project group)
5. Individual
   (experience, talent, constitution of individual programmer)
6. Other

AD-A075 228 BATTELLE COLUMBUS LABS OH F/G 5/2
THE STATE-OF-THE-ART IN SOFTWARE ERROR DATA COLLECTION AND ANAL--ETC(U)
JAN 78 R THIBODEAU DAAG29-76-D-0100
UNCLASSIFIED NL

2 OF 2
AD-
A075228

END
DATE FILMED
5-80
DTIC

Title: Toward a Theory of Test Data Selection, IEEE Trans on SW Eng, Vol. SE-1, No. 2, June 1975

Authors: Goodenough, John B., and Gerhart, Susan L.

Sponsor: U. S. Army, Frankford Arsenal (Phila)

Contract: DAAH25-74-C0469

Author Affiliation: Softech, Inc., Waltham, Mass 02154
Gerhart now at Depart of Comp. Sci, Duke Univ., Durham, N. C. 27706

Discussion:

Types of Programming Errors

Performance Errors (failure to produce results within specified time or space limitations)

Logic Errors (production of incorrect results independent of time and space required)

Construction Errors (failure to satisfy a specification *through error in an implementation*)

Specification Errors (failure to write a specification that correctly represents a design)

Design Errors (failure to satisfy an understood requirement)

Requirements Errors (failure to satisfy the real requirement)

Sources of Errors (manifestation of logic errors as an improper effect produced by an implementation)

Missing Control Flow Paths. Arises from failure to test a particular condition resulting in execution (or non-execution) of inappropriate actions. Ex: failure to test for zero division before executing a divsion.

Inappropriate Path Selection. Occurs when a condition is expressed incorrectly resulting in an action performed (or omitted) under inappropriate conditions. Ex: writing IF A instead of IF A AND B. This results in wrong path if B is false.

Inappropriate or Missing Action. e.g., calculating a value using a method that does not give the correct result (W*W instead of W+W). Failing to assign a value to a variable, calling a function or procedure with the wrong argument list.

Title: Quantitative Aspects of Software Validation, IEEE Tran on SW Eng., Vol. SE-1, No 2, June 1975

Authors: Rubey, Raymond J., Dana, Joseph A. and Biche' Peter W.

Sponsor: Not stated

Author Affiliation: Rubey, Logicon, Dayton, Ohio 45432
Dana, Process Syst Div, Logicon, Merrifield, Va
Biche', Logicon, Torrence, Calif

Discussion:

"... paper discusses the need for quantitative descriptions of software errors and methods for gathering such data." p 150

Two approaches to quantitatively describe the characteristics of SW errors:

1. Actual SW development and validation efforts observed.
2. Create artificial programming environment and develop and observe benchmark programs

Paper includes both types.

Error Classifications: (by type of computer):

- arithmetic
- logic
- processing
- documentation

Phase of SW Cycle in which made

- defining program specs
- designing program
- coding
- maintenance

Errors found in development versus found in validation.
98% of errors found in development.
"..the cost of validation may be disproportionate to the errors left to be found."

Errors found during validation.

Categories

- Incomplete or erroneous specification
- Intentional deviation from specification
- Violation of programming standards
- Erroneous data accessing
- Erroneous decision logic or sequencing
- Erroneous arithmetic computations
- Invalid timing
- Improper handling of interrupts
- Wrong constants and data values
- Inaccurate documentation

Subcategories

- Incomplete or erroneous specification
  - Dimensional error
  - Insufficient precision specified
  - Missing symbols or labels
  - Typographical error
  - Incorrect hardware description
  - Ambiguity in specification or design

Erroneous Data Accessing

- Fetch or store wrong data word
- Fetch or store wrong portion of word
- Variable equated to wrong location
- Overwrite of data word
- Register loaded with wrong data

Erroneous Decision Logic or Sequencing

Erroneous Arithmetic Computations

Paper is based on analyses of actual data but does not indicate where or how these data were obtained.

Title: Measurement, Estimation and Prediction of Software Reliability, NASA CR-145135, Jan 1977

Author: Hecht, Herbert

Sponsor: NASA Cont. No. NAS1-14392

Author Affiliation: Aerospace Corp., Advanced Programs Div, El Sequndo, California

Discussion:

(Ref P4)

Reliability Measurement: The SW is operated over a period of time. Segments of the operation are scored as success or failure by qualitative criteria. From these scores, a single indicator of measured reliability is generated.

Reliability Estimation: Take reliability measurements, as above, on an existing program and modify the result to represent the reliability in a different operating environment. Estimation requires some quantifiable relationships between the measurement environment and that of the estimate.

Reliability Prediction: Any statement about the reliability of a program not based on measurement taken on that particular program. Prediction is normally based on program length, complexity and environmental requirements. One way to measure reliability (p 12) is to observe the operation of the program for some period of time. When no failure is observed in the time period specified, 1/3 failure may be arbitrarily assigned.

Failure rates determined during tests (p 19) are pessimistic estimators of the actual failure rate because the SW is being stressed.

Great simplification in the expressions for failure rates are obtained (p 25) if we assume

a) Every SW failure results in the removal of an error and

b) No new errors are introduced when correcting errors

∴ the correction rate is equal to the failure rate

$$dE(dt = kE(t) \qquad u = \frac{1}{MTBF} = \text{failure rate}$$
$$E(t) = C_o\, e^{-kt}$$
$$U(t) = KE(t)$$

Some problem is noted in the fact that error rates should usually be stated in terms of operating time, but actual results are usually available only as calendar time.(p 26)

Cites TRW study of reliability for USAF using 88 SW routines and 22 variables affecting reliability. (Wolverton & Schick C1971) which showed that no. of reported errors depend mostly on no. of instructions.

SPR = 2.14 + 0.00672 X No. Inst.

SPR = Syst. Prob. Rept.

Failure Classification (p 33)

3 descriptions necessary for meaningful interpret. of SW failure data:

1. time in SW life cycle failure occurred
2. manifestation of failure
3. cause of failure

Time of failure:

1. initial debug
2. test & integration by developer
3. post development test
4. operation

Manifestation of failure:

1. abort of SW system
2. abort of application program
3. persistent gross output errors
4. Temporary gross output errors
5. inaccurate output
6. other

Cause of failure:

1. Specification error
2. Conceptual error in implementing spec.
3. Algorithmic errors (insuff. accuracy or neglect of singularities)
4. Logic & control errors
5. Exceed constraints (time, memory, etc.)
6. Coding errors
7. Data structure errors

Title: Industrial Practices to Control Computer Program Costs., SAMSO-TR-75-293, Jan 1976

Author: Callender, E. D.

Sponsor: Space and Missile Systems Organization, (SAMSO) Air Force Systems Command, L.A., Calif. 90045

Contract: F04701-75-C-0076

Author Affiliation: Aerospace Corp.

Discussion:

The most difficult part of SW development is to ensure that the final product runs correctly. This implies that the sponsor is able to state what the SW is supposed to do when he commissions it. It suggests the ability of the developer to capture the intent of the user.(p 9)

Two techniques for developing good specs (p 9):

1. Management techniques* (config. mgmt)
2. Problem statement languages

Specification language. (need arises from "imprecision in the English language".) It is a stylized format in which the functional specifications can be written by the user.

a) Problem statement language developed by University of Michigan
b) User requirements language developed by USAF

---

* Threads developed by CSC
SW developed under Baseline Configuration Management developed by TRW

Evaluation Criteria for Proposed specification development methods (p 14)

1. Commitment of project management to particular method
2. Ability of system to provide good visibility to mgmt
3. Willingness of management to devote approximately one-third of total cost of system for functional specs and initial design and to wait for code.
4. Ability of system to quickly & easily accommodate changes in specs.

Computer Program Verification (p 25):

Iterative process for determining whether the product of each step of the development process fulfills all requirements levied by the previous step. Eg code verification is comparing implemented code against design specs for correctness. (assumed specs are correct).

Computer Program Validation:

The test and evaluation of the complete program aimed at ensuring compliance with functional specs and criteria. A test of how well program compares with real world from which gem of idea imbedded in the SW was obtained.

V&V should be done by a group in a different organization from the development group and should be allocated 30 to 100% of the development resources (p 27). For contracted development should let separate contract for V&V.

There is no hard evidence for comparing effect of development method on SW reliability.

A good management configuration technique allows feedback of errors found to development group.

Number of semiautomated test tools are evolving.

1. Dynamic analysis coverage programs (p 28):
   - PET
   - RXVP
   - NBS Analyzer
2. Design Assertion Consisting Checker (DACC), TRW
3. Aerospace Corp technique dev for Titan III
   (Code Comparator System)

No indication in this report of any quantitative data base on SW development and resulting errors. In fact P 31 describes an informal procedure for SW development that does not even rely on specs., etc.

APPENDIX C

REPRESENTATIVE PROBLEM REPORTS AND MODIFICATION NOTICES

INCIDENT REPORT

For use of this form, see USACSC REG 18-21-1; the proponent agency is Quality Assurance Directorate. (Test & Conf Mgt Div)

1A. INSTALLATION _____
B. ORIGINATOR NUMBER _____
C. ORIGINATOR LEVEL: ☐
D. OPERATING ENVIRONMENT: ☐☐
E. CONTACT _____
F. CONTACT'S PHONE _____

2A. CAO ID _____
B. FUNCTIONAL AMIS _____
C. DATE RECEIVED _____
D. TIME RECEIVED _____
E. RECEIVED BY _____
F. PHONE _____

G. CONTACT WILL BE AVAILABLE _____
H. PERSON PHONING IN INCIDENT _____ PHONE _____

3. OPERATING STATUS AT TIME OF INCIDENT
A. WAS THERE A CYCLE HALT OR ABNORMAL TERMINATION IN THE PRODUCTION RUN:
☐ YES ABEND CODE
☐ NO
B. DATE/TIME INCIDENT DETECTED:
_____

4. BASELINE _____ INSTL'D _____
A. PROB PGM ID _____ VERS _____
OR
B. PROB DOC ID _____
CHG LEVEL _____ PARA(S) _____
_____ PAGES _____

5. DESCRIPTION OF INCIDENT.
A. ID OF RUN IN PROGRESS _____
B. NARRATIVE DESCRIPTION OF INCIDENT AND RELATED EVENTS _____

C. ACTION TAKEN BY USER AND RESULTS

D. IMPACT ON USER UNTIL RESOLVED

E. CAO/ASD REDEFINITION OF PROBLEM IF DIFFERENT THAN 5B.

6. TYPE OF DIFFICULTY		7. SUPPORTING DOCUMENTATION FOR RESEARCH BY USER			
		AVAIL		AVAIL	
	APPLICATION SOFTWARE		CORE DUMP		OUTPUT FILE-TAPE
	EXECUTIVE SOFTWARE		CONSOLE COPY		DATA FILE-TAPE
	INPUT DATA		JOB CONTROLLIST		RPT OUTPUT-TAPE
	DOCUMENTATION		SYSLST OUTPUT		DOCUMENTATION
	HARDWARE		INPUT FILE-TAPE		OTHER _____
	OTHER		DISK MODULE		

USACSC FORM 53 1 FEB 76

Replaces edition of 1 May 75 which is obsolete

ORIGINATOR NUMBER _____ INCIDENT REPORT

8. CAO DISPOSITION

A. ☐ CAO RESOLUTION PROVIDED AS SHOWN IN BLOCKS 9 THROUGH 11 BELOW.

B. ☐ INCIDENT CANNOT BE RESOLVED BY CAO. FORWARDED TO ASD FOR ACTION AND RESOLUTION ON _____ (DATE/TIME) . VIA ☐ PHONE ☐ TWX ☐ TELECOPY ☐ MAIL ☐ COURIER ☐ OTHER _____

9. METHOD OF RESOLUTION

A. ☐ USER REQUESTED CANCELLATION (DESCRIBE BASIS FOR CANCELLATION IN COMMENTS BELOW)

B. ☐ USER PROBLEM IS A DUPLICATE OF IR/SCR _____ DATED _____

C. ☐ CUSTOMER ASSISTANCE FURNISHED (DESCRIBE ASSISTANCE IN COMMENTS BELOW)

D. ☐ CONVERTED TO EMERGENCY SCR _____ PROJECTED FOR INCLUSION IN EUCP _____ RELEASE DATE O/A _____

E. ☐ CONVERTED TO URGENT SCR _____ PROJECTED FOR INCLUSION IN SCR _____ RELEASE DATE O/A _____

F. ☐ USER IS REQUESTED TO SUBMIT A ROUTINE SCR IAW AR 18-1

G. ☐ COMMENTS. _____

10. REVIEW

A. IN RESOLVING THIS INCIDENT CLOSURE WAS DISCUSSED WITH _____ (NAME) OF _____ (MISO/DPI) ON _____ (DATE) AT _____ HOURS.

B. ITEM ____ OF TABLE B-1 (APP B, CSCR 18-21-1) BEST CATEGORIES THE BASIC CAUSE OF THIS INCIDENT. THE SPECIFIC CAUSE WAS _____

_____ _____ (ANALYST) ☐ CAO ☐ ASD

11. CLOSURE

CAO LOG CLOSED _____ BY _____ CONFIRMING MESSAGE _____

* SYSTEMS CHANGE REQUEST (SCR)

1. TO:	2. FROM:	3. ORIGINATOR NO: 4. POINT OF CONTACT:
5. CATEGORY (CHECK ONE): ☐ EMERGENCY ☐ ROUTINE ☐ URGENT ☐ PRIORITY	6. SUBSYSTEM _____ PROGRAM ID _____ VERSION NO _____	7. INCIDENT ENCOUNTERED STATION _____ DATE _____ TIME _____
8. SHORT TITLE. (25 CHARACTERS MAXIMUM INCLUDING SPACES).		

9. DOCUMENTATION IDENTIFICATION

A. ORI USER MANUALS	☐	C. EXECUTIVE SOFTWARE	☐
B. FUNCTIONAL USER MANUALS	☐	D. FUNCTIONAL SOFTWARE	☐

10. ATTACHMENTS

A. MAPS	☐	D. FILE PRINTOUTS	☐	G. OUTPUT LISTS	
B. CORE DUMPS	☐	E. CONSOLE SHEETS	☐	H. JOB STREAM SEQ.	
C. IMPACT STATEMENT	☐	F. DFSR	☐	I. OTHER	

11. NARRATIVE:

A. PROBLEM DESCRIPTION:

B. RECOMMENDED SOLUTION/ACTION TAKEN:

12. COPY FURNISHED:	13. PREPARED BY:
DATE:	SGNR: _____ DATE: _____

14. PROPONENT AGENT REVIEW:

A. TYPE OF CHANGE	B. CLASS OF CHANGE	C. EXTENT OF CHANGE
☐ FUNCTIONAL	☐ REGULATORY	☐ MAJOR
☐ TECHNICAL	☐ NON-REGULATORY	☐ MINOR

D. REFERRED TO ARA FOR ANALYSIS (DATE): _____

E. DISPOSITION:

☐ APPROVED. REQUESTED IMPLEMENTATION: _____

☐ DISAPPROVED

F. FUNCTIONAL GUIDANCE: ☐ ATTACHED ☐ NOT REQUIRED

☐ TO BE PROVIDED BY _____

SIGNED: _____ DATE: _____

DA Form 4157-R, 1 Feb 76

SYSTEM CHANGE REQUEST ANNEX

ORIG. NO:

For use of this form, see USACSC REG 18-21-1; the proponent agency is Qual Assur Dir (Test & Conf Mgt Div)

ASD — IMPACT ANALYSIS

1. REVIEW & ANALYSIS ____ MAN HOURS
2. DESIGN ____ MAN HOURS
3. PROGRAMING ____ MAN HOURS
4. TESTING ____ MAN HOURS
5. DOCUMENTATION ____ MAN HOURS
6. TOTAL ____ MAN HOURS

7. COMMENTS: Cmd & Staff Updates ____
   a. RECM'D BROADCAST DATE ____
   b. PRIMARY IMPACT:
      PROGRAM ID ____
      VERSION NUMBER ____
8. ESTIMATED MACH. HOURS: ____

HQ USACSC — WORK AUTHORIZATION

☐ APPROVED FOR WORK:

TARGET BROADCAST DATE: ____ WORK PRECEDENCE ____

☐ DISAPPROVED - CLOSE LOG AND NOTIFY ORIGINATOR

COMMENTS:

____ SIGNATURE ____ OFFICE SYMBOL ____ DATE

ASD — SCR CLOSE-OUT DATA

1. BROADCAST VIA EUCP ____ DATE ____ AND/OR
2. BROADCAST VIA SCP. ____ DATE ____ OR
3. CANCELLED PRIOR TO BROADCAST BY ____ DATE ____
4. NOTIFIED BY ____ DATE ____ THAT BROADCAST DID NOT RESOLVE SCR. NEW SCR No. ____ ASSIGNED.
5. REMARKS:

____ SIGNATURE ____ DUTY PHONE ____ DATE

Legend — WORK ASSIGNMENT PRECEDENCE

1. THIS SCR HAS COMMAND ATTENTION FOR EARLIEST POSSIBLE BROADCAST.
2. IMMEDIATE CORRECTIVE ACTION IS REQUIRED TO RESTORE USER SERVICE (Emergency SCR).
3. THIS URGENT (OR PRIORITY) SCR REQUIRES EXPEDITIOUS PROCESSING.
4. TARGET BROADCAST DATE SHOWN MUST BE MET TO SATISFY PA REQUIREMENTS.
5. THIS URGENT OR ROUTINE SCR SHOULD BE INCLUDED IN THE NEXT SCHEDULED SCP.
6. THIS SCR HAS LOWER PRECEDENCE AND MAY BE DEFERRED TO A DOWNSTREAM SCP.

USACSC FORM 8 — [illegible] — To be used with DA Form 4157-R

DATE OF SUBMITTAL: _____ PROJECT IDENTIFIER: _____

SPR IDENTIFIER: _____

SPR TYPE: FR ☐ DC ☐ MC ☐ EF ☐ ED ☐ EC ☐

EMPLOYEE IDENTIFIER: _____

DATE OF PROBLEM DISCOVERY: _____ TIME OF DAY: _____

WORK IDENTIFIER IN PROGRESS: _____

STATUS: OPEN ☐ CLOSED ☐ PENDING ☐ OTHER ☐

PRODUCTS IMPACTED BY PROBLEM: _____

_____

PRODUCTS USED WHEN PROBLEM OCCURRED:

Data Base Identifier: _____

Test Case Identifier: _____

Test Tool Identifier: _____

PROBLEM DESCRIPTION: _____

_____

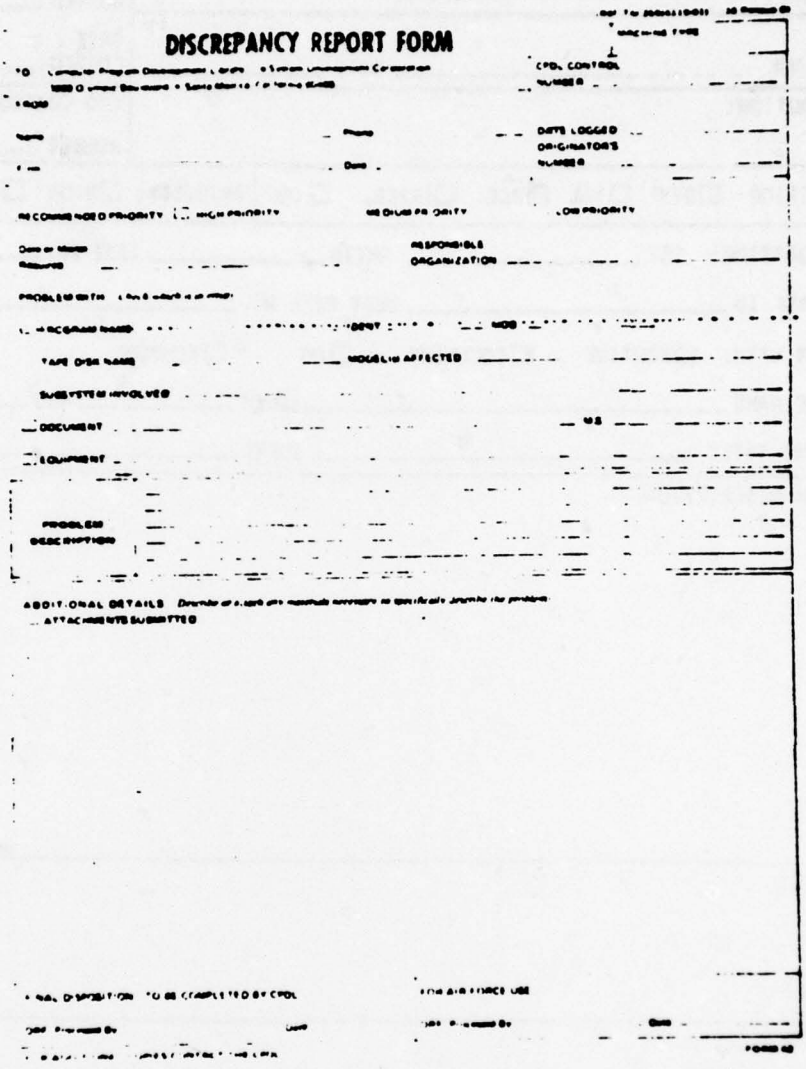
_____

_____

_____

_____

DATE RECEIVED: _____ EMPLOYEE ASSIGNED: _____



DISCREPANCY REPORT FORM

TO

FROM

Name

Phone

Date

CPDL CONTROL NUMBER

DATE LOGGED

ORIGINATOR'S NUMBER

RECOMMENDED PRIORITY

HIGH PRIORITY

MEDIUM PRIORITY

LOW PRIORITY

Date or Month Required

RESPONSIBLE ORGANIZATION

PROBLEM WITH

PROGRAM NAME

REPT

MOD

TAPE DSK NAME

MODULES AFFECTED

SUBSYSTEM INVOLVED

DOCUMENT

PARA

EQUIPMENT

PROBLEM DESCRIPTION

ADDITIONAL DETAILS

ATTACHMENTS SUBMITTED

FINAL DISPOSITION TO BE COMPLETED BY CPDL

FOR AIR FORCE USE

Date

Date

SPR	TO: CONFIGURATION MANAGEMENT	SCMO CONTROL NUMBER
FROM: NAME _____ DATE _____		DATE LOGGED
DISTRIBUTION:		CMO CONTROL NUMBER

TEST PERIOD: ☐DEV ☐VAL ☐ACC ☐INTEG. ☐OO PRIORITY: ☐HIGH ☐MED ☐LOW

CONFIGURATION: SST _____ AMTID _____ TEST ID _____

DATA BASE ID _____ DBRT REEL NO. _____

PROBLEM WITH: ☐ROUTINE ☐DOCUMENT ☐DB ☐COMPOOL

ROUTINE NAME _____ WID _____ MOD _____

DOCUMENT TITLE _____ IDENT _____

PROBLEM DESCRIPTION:

EFFECTS:

COMMENTS:

DEVELOPMENT SOLUTION:

MTM NO _____ CCR NO _____ DBCR NO _____ DUT NO. _____

ROUTINE _____ MOD _____

REMARKS:

PUNCH ALL CARDS

0000

DISCREPANCY REPORT

DISCREPANCY

DISPOSITION

RETEST

SYSTEMS 4614 REV. 8-74

DATA | TO CONFIGURATION MANAGEMENT PAGE 1 of _____

FROM
NAME _____ DATE _____

SCR CONTROL
NUMBER _____
DATE LOGGED ___ ___

MODIFICATION SUBMITTED: SUBSYSTEM ___ _____ _____
PROGRAM NAME _____ MID ___ . ___ . ___ PDU _____
ORIG. CONTROL NUMBER

DOCUMENT TITLE _____ IDENT. NO _____

RESPONSE TO SPR(S) _____ SUPERCEDES MIM _____

REASON FOR SUBMITTAL: ☐ DELIVERY ☐ SPR EXPLANATION ☐ SPR STATUS

TYPE OF DELIVERY ☐ PDR CAPABILITY ☐ MIM CAPABILITY ☐ V&L CAPABILITY (DISS)

☐ SPR FIX ☐ CODING ☐ DATABASE ☐ CC POOL COMPATIBILITY

PROBLEM ANALYSIS: _____

CHANGE DESCRIPTION: _____

_____ ANALYST _____ DATE _____

CHANGE IMPACT:

☐ NO PROBLEM CCR NO. _____ DSCR NO. _____ DUT NO. _____ ECP NO. _____

SUBMITTAL INSTRUCTIONS FOR PROGRAM DELIVERIES:

INPUT MID (VERSION) _____ PDU _____ SOURCE TYPE _____ TAPE NO. _____ REEL NO. _____

VERIFICATION / AUTHORIZATION:

MIM APPROVED BY _____ DATE _____ CONFERENCE _____ DATE _____

DISPOSITION:

ENCOD AMT _____ BY _____ DATE _____ VERIF. BY SCN _____

TR

ORIGINATED BY	EXT.	ROOM	DATE	TR NUMBER
PIDENT (Program Identification)		LEVEL	PRIORITY E U R	REASON FOR CHANGE ☐ ERROR ☐ IMPROVEMENT
COMPUTER RUN	TYPE OF ACT.	TEST ID	DUMP TAPE NO.	LOG TAPE NO.
ORIGINATION APPROVAL (Supervisor)		DATE	RESPONSIBLE PA/PA (Manager)	DEPT
DESCRIPTION				
SCM LOG-IN (Coordinator) DATE		APPR. FOR ACTION DATE	REFERRED TO	DEPT

CR

ORIGINATED BY	EXT	ROOM	DATE	CR NUMBER
PIDENT		LEVEL	LIBRARY ID	VOLUME SER. NO.
DESIGN APPROVAL		DATE	BELLFLOW COMMENTS UPDATED (Flowchart) ☐ YES ☐ NO ☐ NA	
DESCRIPTION				
CHANGE RECEIVED		DATE	CHANGE INTEGRATED	DATE
RELEASED IN:		PIDENT		LEVEL

1-72

PAGE 1

Column	Description
1- 7	Routine that fix was made to
8-11	SPR No.
12-18	Routine that error was found in
19-26	Date SPR opened
27	Criticality category
28-29	Test period
38-42	Error category code
49-52	X in appropriate column to indicate what type of fix was involved
53-54	EX in these columns if fix was explanatory or not to a routine
55-62	Date SPR closed
63-68	Version of project for which SPR applies
69-70	Project number
71	X indicates SPR documents more than one problem
72-74	No. of days SPR opened

Columns	Field	Code
1	File Identification	"2"
2-6	Project Identification	Alphanumeric
7-8	Project Code	Alphanumeric
9-12	SPR Number Right justified Blank if no SPR#	Numeric
13-19	Module Affected Identification Left justified	Alphanumeric
20-21	Version Identification	Alphanumeric
22-29	Date SPR Opened (MM/DD/YY) Blank if no SPR	Alphanumeric
30	Termination Code Blank = Terminated Normally S = Software Aborted H = Hardware Aborted	Alphabetic
31	Seriousness of Problem 1 = Critical 2 = Medium priority 3 = Low priority 4 = Suggested important	Numeric
32	Test Period D = Development - Unit Test V = Validation - Unit Acceptance I = Integration A = Acceptance of Build O = Operational Demonstration	Alphabetic
33-37	Error Category	Alphanumeric
38-41	Applicable SMN Number	Numeric
42-46	Type of Correction New Module Update X in Col 42 Document Update X in Col 43	Alphabetic

Columns	Field	Code
42-46	COMPOOL Change X in Col 44 Data Base Change X in Col 45 Explanation X in Col 46 Leave column blank if not applicable. Use more than one type if several apply.	
47-54	Date SPR Closed (MM/DD/YY) The SPR is closed by an SMN, therefore, this data is taken from the SMN.	Alphanumeric
55-57	Days Open Total of working days between the open and closed date. If only an SMN appears it reflects 1 day open. Right justified.	Numeric


FILMED
5-8